ARCHIVOS

El subsistema de archivos o sistema de gestión de archivos es el sistema de software que proporciona a los usuarios y aplicaciones los servicios relativos al uso de archivos.

Sus objetivos son:
- ü Cumplir con la gestión de datos y solicitudes del usuario.
- ü Garantizar la integridad del contenido de los archivos
- ü Dar soporte de E/S para los distintos dispositivos
- ü Brindar un conjunto de rutinas standard de interfaces de E/S.
- ü Optimizar el rendimiento tanto a nivel de productividad como de tiempo de respuesta.

Desde el punto de vista del usuario, este debe poder crear, borrar y modificar sus archivos; controlar su acceso y el de los otros usuarios; definir que tipo de acceso utilizará; mover datos entre sus archivos; hacer backups y poder recuperar la información desde ellos; etc.

Definiciones de archivo

- ç es una colección de datos
- ç es un conjunto de datos relacionados lógicamente

El usuario identifica a un archivo a través de un nombre.
El archivo reside en memoria auxiliar (discos, cintas, diskettes, etc).

El sistema operativo se abstrae de la estructura física para crear una estructura lógica: el archivo. Es a través del sistema operativo que los archivos se almacenan en dispositivos físicos, y por lo tanto, un subsistema de él se encarga de la administración.

Tipos de archivos

Los archivos representan datos y programas. Los archivos de datos pueden ser numéricos, alfabéticos, binarios. Un archivo es una secuencia de bits, bytes, líneas, registros, de acuerdo a lo que necesita el usuario creador de ese archivo.

Hay, entonces, diferentes tipos de archivo:
Archivo de texto (Text file): es una secuencia de caracteres organizados en líneas.
Archivo fuente (Source file): es una secuencia de rutinas y funciones donde cada uno de ellos es una secuencia de declaraciones seguido de sentencias ejecutables.
Archivo objeto (Object file): es una secuencia de bytes organizadas en bloques que comprende el enlazador (linker).

Archivo ejecutable (Executable file): es una secuencia de bytes organizadas en secciones de código que el cargador (loader) pone en memoria y ejecuta.

El sistema operativo para trabajar de una manera razonable con un archivo debe saber que tipo de archivo es. Imprimir un archivo binario o mostrarlo por pantalla, nos desplegará un serie de signos no legibles.

Por eso es importante que se almacene junto con la información relacionada con el archivo qué tipo de archivo es.
En UNIX existe el comando file que permite saber con que tipo de archivo estoy trabajando. Antes de imprimir, por ejemplo, el archivo Xmlklo, me aseguro que sea un tipo legible emitiendo el comando

file Xmlklo

si es un ASCII puedo verlo o imprimirlo.

Atributos de un archivo

Los atributos de un archivo son: el nombre, el tipo, la localización (donde se ubica), derechos de acceso, tiempo de creación/acceso/modificación, UID del creador, etc.
Podemos también citar como características las siguientes:

- ü Volatilidad, que es la frecuencia con la que se agregan y borran ítems en un archivo;
- ü Actividad, que es el porcentaje de ítems accedidos durante un determinado período de tiempo;
- ü Medida, que es la cantidad de información almacenada en el archivo

Operaciones

Podemos pensar en un archivo como en un tipo abstracto de datos, al que se pueden aplicar las siguientes operaciones:

- ¢ Abrir
- ¢ Cerrar
- ¢ Crear
- ¢ Destruir
- ¢ Copiar
- ¢ Renombrar (cambiar el nombre)
- ¢ Listar

Los ítems que forman el archivo (bytes, registros, etc.) pueden:

- ¢ Leer
- ¢ Modificar
- ¢ Agregar (al final)
- ¢ Insertar
- ¢ Borrar

Como veremos estas operaciones son la composición de unas pocas, implementadas a través de llamadas al sistema (systems calls: open, close, read, write, seek, status, etc.).

Filesystem

Filesystem se traduce como "sistema de archivos". Es la forma en que dentro de un sistema de computo se organizan, se administran los archivos.
Esa administración comprende:

- Métodos de acceso: cómo se acceden los datos contenidos en el archivo
- Manejo de archivos: cómo actúan los mecanismos para almacenar, referenciar, compartir y proteger los archivos
- Manejo de la memoria secundaria: Cómo se administra el espacio para los archivos en memoria secundaria.
- Mecanismos de integridad: con qué métodos se garantiza la incorruptibilidad del archivo (que esté la información que tiene que estar y la que no debe estar ahí, que esté en otro lado).

Cada disco en un sistema de computo contiene como mínimo una partición que es una estructura de bajo nivel donde residen archivos y directorios. Las particiones pueden separar áreas dentro de un disco.
En algunos casos se puede necesitar una estructura que permita que un archivo se extienda en mas de un disco. En este caso se crea una estructura lógica que permite que el usuario trabaje una abstracción de la estructura de disco real. El sistema operativo ofrece al usuario un espacio que es la suma de la capacidad de almacenamiento de los discos reales. Por ejemplo: si hay 2 discos de 1 Gb, el usuario vera un espacio de 2 Gb (pudiendo crear estructuras incluso de 1 Gb y medio) y el sistema operativo se encargará de la administración sobre los discos reales.

En el siguiente ejemplo contamos con tres discos (1,2 y 3). En el disco 1 se definió la partición A y la B, cada una de ellas con su directorio y archivos.
La partición C en cambio, ocupa dos discos, el 2 y el 3.

El usuario final puede llegar a trabajar con los archivos sin conocer ni en qué disco ni en qué partición están, si maneja una buena abstracción.

<table>
<tr><td rowspan="2">Partición A</td><td>directorio</td><td rowspan="4">Disco 1</td></tr>
<tr><td>archivos</td></tr>
<tr><td rowspan="2">Partición B</td><td>directorio</td></tr>
<tr><td>archivos</td></tr>
<tr><td rowspan="2">Partición C</td><td>directorio</td><td>Disco 2</td></tr>
<tr><td rowspan="2">archivos</td><td>Disco 3</td></tr>
</table>

Cada partición tiene información sobre los archivos dentro de ella. Esa información se mantiene en un directorio del dispositivo (*device directory*), o en una tabla de contenido del volumen (volume table of contents, *vtoc*, en algunos sistemas). Normalmente, en los directorios se ubica toda o parte de la información sobre el archivo.

Directorio

El directorio es como una tabla que traslada los nombres de archivo en la referencia necesaria para ubicar el archivo dentro de la estructura

Normalmente el directorio tiene información sobre los archivos que pertenecen a él, tal como atributos, ubicación o propietario.
Las rutinas del sistema son las que acceden a esta estructura y proporcionan la información del directorio a usuarios y aplicaciones.

Cada sistema organiza esta información a su manera: algunos construyen una cabecera (*header* del archivo) y la separan del directorio, quedando información primaria en el directorio para localización del archivo y el resto en el header. Otros estructuran el directorio como una simple lista, con una entrada por archivo, como si fuera un archivo secuencial donde el nombre de cada archivo es la clave.

Las operaciones que se pueden realizar en un directorio son:

- Buscar un archivo
- Crear un archivo
- Borrarlo
- Listar el directorio
- Cambiar el nombre del archivo
- Navegar por el filesystem

Directorio de un nivel: Todos los archivos están contenidos en un directorio. Fácil de soportar y entender. Se complica cuando hay muchos archivos y mas de un usuario. Los archivos deben tener nombres únicos.

Directorio de dos niveles: En esta solución hay un directorio para cada usuario. Cada usuario tiene su UFD (user file directory) donde cada UFD tiene una estructura similar. Existe un MFD (master file directory) que esta indexado por nombre de usuario o cuenta y que apunta al UFD del usuario.

Al crear un archivo se testea la UFD del usuario para ver si ese nombre ya existe. Cuando un usuario quiere borrar un archivo que tiene determinado nombre, no hay forma de que borre uno que se llame igual en el directorio de otro usuario: sus actividades se limitan a su UFD.

El problema de esta estructura es que aísla un usuario de otro, en el caso de que estos necesiten cooperar, y tener archivos comunes.

Estructura de árbol: es una generalización de los niveles vistos. El usuario puede crear sus propios subdirectorios y así, organizar sus archivos como le plazca.
Cada usuario tiene un directorio inicial de login, que es el directorio donde queda el usuario cuando se conecta. Tiene comandos para ir moviéndose por la estructura (cd, change directory).
Ejemplos: Estructuras de directorio

En CP/M hay (había) un solo directorio.

Código Usuario	Nombre Archivo	Tipo	Grado	Conteo	Nros de bloques de disco
1 byte	8 bytes	3 bytes	1 byte	1 byte	16 bytes

Grado indica cuantas entradas toma el archivo, si tiene mas de 16 bloques.
Conteo indica cuantas de las 16 entradas del bloque están en uso.

En MS/DOS, la entrada del directorio tiene 32 bytes de largo.

Nombre	Extensión	Atributos	Reservado	Tiempo	Fecha	1er nro. bloque	Tamaño
8 bytes	3 bytes	1 byte	10 bytes	2 bytes	2 bytes	2 bytes	4 bytes

En UNIX System V, la estructura del archivo directorio es una tabla donde cada entrada tiene esta estructura:

Numero de inodo	Nombre del archivo
2 bytes	14 bytes

Arquitectura del software de un sistema de archivos

<table>
<tr><td colspan="4">Programa de usuario</td></tr>
<tr><td>Pila</td><td>Secuencial</td><td>Secuencial Indexado</td><td>Indexado</td></tr>
<tr><td colspan="4">E/S lógica</td></tr>
<tr><td colspan="4">Supervisor básico de E/s</td></tr>
<tr><td colspan="4">Sistemas de Archivos Básico (nivel de E/s física)</td></tr>
<tr><td colspan="2">Manejador del Disco</td><td colspan="2">Manejador de la cinta</td></tr>
</table>

Nota: Stallings llama pila a la organización de archivos donde los datos se recogen en la forma que llegan. O sea, acumula una cantidad de datos y los guarda. Puede haber campos diferentes, o similares en distinto orden. Cada campo debe ser autodescriptivo y debe predeterminarse cuales son los delimitadores.

Los ***manejadores de dispositivo (device drivers)*** se comunican directamente con los dispositivos o sus controladores o canales.
Los manejadores de dispositivos tienen la función de comenzar las operaciones de E/S en un dispositivo y procesar la terminación de una solicitud de E/S.

El ***sistema de archivos básico*** trata con bloques de datos que son los que se intercambian con los discos o cintas. Ubica estos bloques en el almacenamiento secundario o en el intermedio en memoria principal. Este sistema normalmente se considera parte del SO.

El ***supervisor básico de E/s*** se responsabiliza de iniciar y terminar la E/s con archivos. Selecciona el dispositivo donde se realizará la E/S, según el archivo seleccionado. Planifica los accesos a disco y cinta, asigna los buffers de E/S y reserva la memoria secundaria. Es parte del SO.

La ***E/S lógica*** tiene la función de permitir a los usuarios y aplicaciones acceder a los registros.

El ***método de acceso*** es el nivel mas cercano al usuario, proporcionando una interfaz entre las aplicaciones y los archivos.

Los usuarios y las aplicaciones tratan con registros y la E/S se realiza de a bloques. Los registros deben ablocarse para la salida, y desablocarse en la lectura. Para manejar la E/S con bloques deben proveerse las funciones para ello: por ejemplo, se debe gestionar el almacenamiento secundario (asignación de bloques libres de memoria secundaria a los archivos, gestionar el espacio libre, manejar la solicitud de bloques individuales. Todo esto debe organizarse optimizando el rendimiento.

Funciones del filesystem

El filesystem debe proveer las herramientas para que los usuarios puedan realizar las siguientes acciones:
Usuarios

- Deben poder crear, modificar y borrar archivos.
- Deben poder compartir los archivos de manera controlada.
- Debe poder organizar su archivo de acuerdo lo exija la aplicación
- Debe poder transferir información entre los archivos.
- Deben poder referirse a sus archivos con nombres simbólicos, en vez de tener que usar los físicos.
- Deben poder resguardar (*backup*) y recuperar (*restore*) la información
- Debe contar con mecanismos para encriptación y de-criptación, para ambientes de gran exigencia de privacidad y seguridad.

El filesystem debe proveerle al usuario una interfaz que haga fácil la interacción con los archivos. El usuario debe contar con una visión lógica de sus archivos, no física.

Designación de un archivo

El usuario se refiere al archivo por su nombre simbólico, que debe ser único para permitir su localización sin ambigüedad.
En un sistema jerárquico, cualquier archivo puede ser localizado siguiendo un camino desde el directorio raíz o maestro, descendiendo por las ramas hasta el directorio donde se ubica el archivo. Este es el nombre del camino *(full pathname)* del archivo.

El directorio asociado con cada usuario es su directorio de trabajo, o de login. Como el usuario trabaja normalmente sobre ese archivo, normalmente se refiere a sus archivos directamente, sin dar el nombre completo: se le llama *relative pathname*.

Normalmente el usuario asigna un nombre de archivo que lo relacione con su contenido. Hay sistemas con restricciones en tipo y cantidad de caracteres, con extensión (.xls, .doc, .bat), con versionado (pepe.cmd;1).

Estructuras asociadas

Hay estructuras que se crean en memoria durante la ejecución de los procesos relacionadas con el uso que estos hacen de los archivos.

El sistema cuenta con una tabla de archivos abiertos (***open file table)*** general donde figura información sobre los archivos activos (que están siendo usados por los procesos) e índices de referencia al archivo. Algunos sistemas requieren hacer un *open* previo al primer *read*.
La llamada a sistema *open* toma el nombre de archivo, busca en el directorio, copia la entrada del directorio en la tabla de archivos abiertos. Todo esto si estas acciones son permitidas de acuerdo a los modos de protección del archivo.
Un puntero para accederlo rápidamente es también puesto en la tabla de archivos abiertos.
En un sistema multiusuario, un archivo puede ser abierto por mas de dos usuarios simultáneamente, cada uno con su puntero.
Hay dos tipos de tablas. La ***tabla por proceso*** contiene información sobre los archivos abiertos por el proceso y contiene un apuntador a la segunda tabla, la de los archivos abiertos en todo el sistema. Esta segunda tabla tiene información independiente del proceso que lo ejecute como ubicación del archivo en el disco, medida del archivo, etc.
La tabla del sistema tiene también la cantidad de procesos que tienen abierto el archivo, y con cada *close* este valor se va decrementando. Cuando queda en cero, se remueve esa entrada.
La información asociada a un archivo abierto es:

- Puntero (File pointer)
- Contador de aperturas del file (File open count) cuando lo tiene abierto mas de un proceso simultáneamente
- Ubicación del archivo en el disco

Hay sistemas que permiten compartir entre varios procesos, secciones de un archivo: un archivo ejecutable que reside en disco y es llevado a memoria y forma parte del espacio de direcciones de varios procesos; o archivos de datos cuyos bloques en memoria son accedidos por varios procesos.

Organización de los archivos

- Secuencial (según el orden físico)
- Directa (acceso random)
- Indexada Secuencial (organización de acuerdo a una clave)
- Particionada: es un archivo de varios subarchivos. Cada subarchivo secuencial es un miembro. La dirección inicial de estos miembros figura en un directorio del archivo. Se utiliza para el almacenamiento de macros o bibliotecas (*libraries* mal llamadas "librerías").

File control block ó header de archivo

Hay un conjunto de información relacionada con el archivo que es necesaria para su manejo. Por ejemplo:

- Su nombre simbólico
- Su ubicación en memoria secundaria
- Organización
- Tipo de dispositivo donde reside
- Control del acceso
- Tipo de archivo
- Fecha de creación
- Fecha de última modificación
- Contadores que dan datos sobre la actividad del archivo

La ubicación de esta información depende de cada implementación. En algunos casos, esta información esta una parte en el *header* y otra en la entrada de directorio correspondiente al archivo.

El *header* se modifica cuando cambian algunas de sus variables, aunque no haya modificación del archivo. Pueden cambiar los controles de acceso (cambiar permisos o atributos) y no modificar el contenido del archivo. En cualquier caso en que se modifique el archivo se modifica la fecha de última modificación, por lo tanto se modificará el *header* también. Aunque simplemente lo lea por pantalla, se modificará la fecha de acceso, por lo tanto, también se modificará el header.

En el momento que el archivo esta siendo accedido es necesario que su *header* esté en memoria para ser accedido rápidamente, y además, porque en algunos casos, contiene las referencias a los bloques de disco que contienen al archivo.

Gestión del espacio libre en disco

La tabla de asignación de disco controla los bloques de disco que están disponibles.

Se utilizan normalmente tres técnicas: tablas de bits, bloques libres encadenados e indexación.

Las tablas de bits usa un vector donde cada componente (bit) indica cada bloque del disco: si es 0, el bloque esta libre; si es 1, esta ocupado.
La ventaja es que es fácil de determinar el estado de un bloque y para fijar su estado, basta modificar el valor del bit. Además, es pequeña y puede mantenerse en memoria.

Los bloques libres encadenados permite tener solamente el puntero al inicio de la cadena. Si necesito un bloque libre, tomo el primero de la cadena, cambiando luego el valor del puntero inicial que ahora señalará al siguiente.

La indexación trata el espacio libre como si fuera un archivo y usa una tabla con índices.

Asignación de espacio en disco

Un archivo al residir en memoria secundaria se transforma en un conjunto de bloques.
El encargado de asignar bloques a los archivos es el SO a través del subsistema de gestión de archivos.
Las tareas comprenden asignar los bloques y llevar un registro de los bloques que están ocupados y de los que no lo están (gestión del espacio libre).

En la asignación se consideran cuestiones tales como si se asigna el máximo de espacio que necesita un archivo de una sola vez, si se asignan espacios contiguos, como se guardará constancia de los espacios asignados.

La tabla de asignación de archivos contiene información sobre como localizar los archivos en la memoria secundaria. Normalmente tiene una entrada por archivo.

Asignación previa y asignación dinámica

Para asignarle a un archivo todo el espacio que necesita de una vez, se tendría que conocer la cantidad exacta de bloques que ocupará. Si bien en el caso de copia o compilación puede saberse de antemano el tamaño, no es lo común, en el caso de archivos de datos.
Si la asignación previa fuera la única alternativa, los programadores terminarían sobredimensionando los archivos para no quedarse sin espacio, lo que conduciría a tener gran cantidad de espacio desaprovechado dentro de los archivos.

En el libro de W. Stallings, se habla del concepto de sección, como unidad de asignación y formado por un conjunto de bloques. Con este concepto, hay 4 elementos que afectarán el rendimiento de la gestión de archivos:

- La contigüidad del espacio mejora el rendimiento pues facilita la operación de recuperación, por ejemplo.
- Muchas secciones pequeñas aumentarán el tamaño de las tablas de gestión.
- Secciones de tamaño fijo, simplifica la asignación de espacio.
- Secciones de tamaño variable o pequeñas de tamaño fijo, ayudan a minimizar la fragmentación.

Las estrategias para asignación de espacio en secciones de tamaño variable son las mismas que vimos al estudiar memoria: first fit, best fit y worst fit.

Asignación contigua, encadenada e indexada

Contigua: En este tipo de asignación, cuando se crea un archivo se le asignan secciones contiguas.
La tabla de asignación de archivos contiene solo una entrada por archivo con el bloque de comienzo y el tamaño del archivo. Es práctico para un archivo secuencial.
Desventajas: produce la fragmentación externa. Habrá que implementar técnicas de compactación. Además se debe declarar el tamaño del archivo en el momento de la creación.

Encadenada o enlazada: La asignación se hace de a bloques individuales, y cada bloque contendrá el puntero al siguiente.
La tabla de asignación necesita una sola entrada por archivo con el bloque de comienzo y su longitud. Cualquier bloque libre puede agregarse a la cadena.
Es práctico para los archivos secuenciales.
Desventaja: para acceder a un bloque individual, debo acceder a los anteriores para hallar la cadena, es decir, no puedo acceder a un bloque individual directamente.
Además, al estar separados los bloques, para traer varios bloques del archivo, se deben recuperar uno por uno.

Indexada: en un bloque aparte se almacenan las direcciones de los bloques. La entrada en la tabla de asignación de archivos apuntará a este bloque.
Elimina la fragmentación externa, soporta el acceso secuencial o directo. Es la mas común.

Ejemplo sobre asignación enlazada

Este método de almacenamiento no contiguo tiene diferentes implementaciones.
Supongamos que contemos con bloques de 1024 bytes (1K). Una forma de almacenamiento sería que de ese bloque de 1024 bytes, 1022 sean para datos y los 2 restantes, indiquen cual es el próximo bloque.
Desventajas:

1. el nro de bytes disponibles no es múltiplo de dos lo que dificulta la aritmética binaria.
2. El acceso al azar es costoso de implementar pues, por ejemplo, para acceder al tercer bloque, debo acceder a los dos anteriores.

Supongamos que quiero leer el byte 32768. Divido 32768 por 1022 (bytes disponibles en el bloque). El cociente es 32 y el resto 64, por lo tanto, el byte 32768 esta en el bloque 33, desplazamiento 64. Para poder accederlo deberé leer los 32 bloques anteriores para poder llegar al 32, de donde leeré la dirección del bloque 33 que es el que busco.

Apuntadores en memoria. Ejemplo de asignación indexada: La FAT

Veamos como es la asignación en MS-DOS.
Supongamos 3 archivos: A, B y C.

A ocupa los bloques: 6, 8, 4 y 2.
B ocupa los bloques: 5, 9 y 12.
C ocupa los bloques: 10, 3 y 13.

En cada disco hay una tabla de asignación de archivos (FAT) que tiene una entrada por cada bloque de disco.
En el directorio de cada archivo esta el número del 1er bloque del archivo (en caso de A, el 6; para B, el 5 y para C, el 10.
El problema con la FAT es que los apuntadores de todos los archivos del disco están en la misma tabla. Aunque se abra solo un archivo, se necesita cargar toda la FAT en memoria.

Nro de bloque	Estado o siguiente
0	Tamaño del disco
1	
2	EOF
3	13
4	2
5	9
6	8
7	FREE
8	4
9	12
10	3
11	FREE
12	EOF
13	EOF
14	FREE
15	BAD

Caso de ejemplo: Administración de archivos en System V, Unix

Este es otro ejemplo de asignación indexada.
Cada archivo de Unix tiene asociado un inodo, nodo-i o header.
La lista de las direcciones de los bloques que forman un archivo se ubican en su inodo. Son 13 direcciones: 10 directas y 3 indirectas.
Supongamos que sean bloques de 1k. Si el archivo tiene 10 K, los bloques del archivo serán accedidos a partir de las direcciones directas que están en el inodo. El inodo esta en disco, por lo tanto, hay que leer primero el inodo y desde allí leer las direcciones para acceder a los bloques.
Cuando crece a un bloque mas, se le asigna un bloque de disco y se coloca un apuntador indirecto.
El bloque asignado es un bloque que tendrá direcciones, y cada una de estas referenciará a un bloque donde están los datos que continuan.

Si las direcciones son de 32 bits, en un bloque de 1024 bytes, entrarán 256 direcciones de disco.
Si uso los 10 directos mas el de una indirección, puedo direccionar 266 bloques de disco (10 directos mas 256 direcciones del indirecto).
Si el archivo mide mas de 266 k, cuando necesito agregar bloques al archivo tendré que usar la dirección de dos niveles de indirección. Esta dirección indica un bloque cuyo contenido son 256 direcciones de 32 bits. Cada una de estas direcciones referencian un bloque cuyo contenido son 256 direcciones de 32 bits. Cada dirección referencia, ahora si, el bloque de datos que agregan al archivo.
De esta manera, con dos niveles de indireccion logro tener un archivo de 10K + 256K + (256*256)K, sumando los bloques indirectos, los de 1 nivel de indireccion y los de dos niveles de indireccion.
Si el archivo mide mas que este tamaño, se debe usar el tercer nivel de indireccion. Esta dirección indica un bloque cuyo contenido son 256 direcciones de 32 bits. Cada una de estas direcciones referencian bloque cuyo contenido son 256 direcciones de 32 bits. Cada dirección referencia un bloque cuyo contenido son 256 direcciones de 32 bits. Ahora si: cada dirección de ese bloque referencia el bloque de datos.

Los bloques indirectos se utilizan solo cuando se necesitan.

Estructura del filesystem en System V

Cada filesystem en Unix System V tiene la siguiente estructura:

1. boot block
2. superblock
3. lista de inodos
4. bloques de datos

El ***boot block*** puede contener el *bootstrap code* que lee la máquina para bootear o iniciar el sistema operativo. Cada *filesystem* tiene el *boot block* aunque este vacío.
El ***superblock*** describe el estado del filesystem:

- ü medida
- ü cantidad de archivos que puede almacenar
- ü cantidad de bloques libres
- ü como ubicar espacio libre en el disco (lista de bloques libres)
- ü cantidad de inodos libres
- ü medida de la lista de inodos
- ü indicador de que el superblock ha sido modificado
- ü etc.

Se especifica la medida de la lista de inodos en el momento de la configuración del filesystem. El kernel referencia inodos a través de un índice en la lista de inodos.
Los bloques de datos comienzan al final de la lista de inodos. Un bloque de datos puede pertenecer solo a un archivo.

Como se accede a un inodo?

El kernel identifica inodos de acuerdo al filesystem al cual pertenece y su número de inodo, y lleva una copia de ese inodo a memoria, a traves de algoritmos de alto nivel.

Conociendo el número de inodo y el dispositivo lógico al cual pertenece realiza un cálculo para ubicar el bloque de disco que lo contiene (para ello debe saber cuantos inodos entran en un bloque de disco).

El calculo es el siguiente:

Nro bloque disco = ((nro inodo –1)/número de inodos por bloque)+ bloque comienzo de la lista de inodos.

Asi, al conocer el número de bloque de disco donde está el inodo, lo puede leer y llevarlo a memoria para su procesamiento.

Como se accede al contenido de un archivo?

Cuando se quiere abrir un archivo, se debe ubicar primero cual es su inodo. Esto se hace accediendo al directorio al cual pertenece el archivo y en la entrada correspondiente a su nombre, esta el número de inodo (como vimos en el ejemplo de estructuras de directorio).
Se accede entonces a ese inodo, y de allí se tomarán las direcciones para llegar al contenido.
Debemos considerar que el directorio es también un archivo, por lo tanto para acceder a él, deberemos buscar en el directorio padre cual es su inodo, leerlo y recién ahí puedo ver el número de inodo del archivo que me interesa.

Protección de la info: el Intercalado de discos

La técnica de intercalado de discos (disk stripping) permite que secciones sucesivas de un archivo se almacenen en discos diferentes de un arreglo (array) o vector de discos. La ventaja de esta técnica es que las peticiones de E/S de un archivo están uniformemente distribuidas por todos los discos del vector, y el sistema se mantiene equilibrado. Las peticiones de E/S al archivo se distribuyen uniformemente mejorando la productividad y el tiempo de respuesta.

Hay diferentes técnicas que dependen de la granularidad del intercalado: las ***unidades independientes*** lo hacen a nivel de bloque; los ***vectores sincronizados*** lo hacen a nivel de byte.

Práctica 6: Archivos

Para ofrecer una imagen única del sistema, el sistema operativo debe ofrecer servicios que permitan asociar, y desasociar, unos sistemas de otros de forma transparente en un árbol de nombres único. Además, las utilidades de interpretación de nombres deben ser capaces de saltar de un sistema de archivos a otro sin que sea aparente en ningún momento el nombre del dispositivo físico o lógico que almacena el sistema de archivos. Las dos llamadas al sistema de Unix que realizan estas operaciones son *mount* y *umount*. La operación de montado permite conectar un sistema de archivos, almacenado en un volumen o partición, a una entrada de directorio del árbol de directorios existente.

Por ejemplo, si en Linux colocamos el comando:

$mount

Nos muestra los volúmenes y particiones montadas en el sistema (la primera columna) y el directorio en donde han sido conectados, denominado "punto de montaje" (la segunda columna), también podemos ver el tipo de sistema de archivos que dicho volumen o partición tiene, por ejemplo: *vfat* para sistemas de archivos de Windows 95 ó 98, *ntfs* para sistemas de archivos de Windows NT, *ext2* ó *ext3* (Extended Second, Extended Third) para sistemas de archivos de Linux. Y además algunas opciones del montado. Por ejemplo: un sistema de archivos *iso9660*, propio de los CD-ROM, estará montado como "sólo lectura" o *read-only*. En cambio otros pueden estar montados para lectura/escritura o *read-write*.

Los comandos *mount* y *umount* tienen muchas opciones, puede verlas en la documentación en línea, con el comando:

$ man mount
$ man umount

Estos comando toman las opciones por defecto del archivo "/etc/fstab" que contiene una descripción de los distintos volúmenes/particiones, con sus respectivos puntos de montaje, tipo de sistema de archivo y opciones adicionales. Véalo con el comando:

$ cat /etc/fstab

Este archivo menciona varias particiones de nuestro disco rígido ¿Cómo podemos verlas? Coloque:

fdisk -l /dev/hda

Y obtendrá un listado de cada partición, si está marcada como "activa" para que pueda arrancar desde ahí (boot), los cilindros que abarca (desde/hasta), y el tipo de partición de

la que se trata. De acuerdo con el tipo de partición será el tipo de sistema de archivos que podrá tener (**Cuidado:** no confundir tipo de partición con tipo de sistema de archivos).

¿Cómo se crea un sistema de archivos? Lo que en la jerga del DOS se llama "formatear" en directa alusión al programa FORMAT que realiza esa operación, puede hacerse en Linux con el comando genérico "mkfs" (es decir, justamente, **m**a**k**e **f**ile**s**ystem) seguido de un punto y el sistema de archivos específico, por ejemplo "mkfs.vfat" ó "mkfs.ext2".

```
# mkfs.vfat    /dev/fd0
```

También podemos usar el comando específico, por ejemplo si queremos crear un sistema de archivos *extended second* (es decir, para Linux) en un diskette, deberíamos colocar:

```
#mke2fs       /dev/fd0
```

En vez de:

```
# mkfs.ext2   /dev/fd0
```

Que también es válido. Si quisiéramos comprobar el sistema de archivo, deberíamos colocar, análogamente:

```
# fsck.ext2   /dev/fd0              (o su equivalente)
# e2fsck      /dev/fd0
```

Si el sistema está marcado como "limpio", será necesario "forzar" la comprobación con la opción "-f".

```
# e2fsck      -f      /dev/fd0
```

Para ver el contenido, es decir, para poder acceder al sistema de archivos que hay en él, es necesario montarlo, como dijimos anteriormente. Por ejemplo:

```
# mount       -t ext2         /dev/fd0       /mnt/floppy
```

Con todas sus opciones, estamos diciendo que el tipo de sistema de archivos es "ext2", que el dispositivo es "/dev/fd0" y que debe ser montado en "/mnt/floppy". Ahora podemos listar el contenido del sistema de archivos del diskette con:

```
$ ls    /mnt/floppy
```

Aunque está vacío, porque acabamos de crearle el sistema de archivos, hay un directorio llamado "lost+found", es decir "perdidos+encontrados", en ese directorio, típico de éste sistema de archivos, se guardan archivos (o porciones de archivos) que se han recuperado luego de una falla (e2fsck los coloca ahí).

Podemos realizar varias operaciones con archivos y directorios, primeramente nos cambiaremos al directorio en el que montamos el diskette, para trabajar sobre él.

cd /mnt/floppy

Acá van una pequeña lista no exaustiva, por ejemplo, para crear un directorio llamado "nuevo": (aunque no es necesario en muchos casos, seguiremos usando la cuenta del superusuario "root")

mkdir nuevo

Para cambiarnos dentro de ese directorio:

cd nuevo

Podemos con el comando "touch" crear un archivo si es que éste no existe, o actualizarle la fecha de acceso en caso de que el archivo exista, por ejemplo:

touch miarchivo

Podemos crear un enlace simbólico con:

ln -s miarchivo mienlace

Veámoslos en formato largo, con color:

ls -l --color

Si intentamos editar el enlace con:

vi mienlace

Podemos agregar el siguiente texto si entramos en modo inserción:
Es este miarchivo o mienlace?

Ahora veamos el contenido de ambos:

cat mienlace
cat miarchivo
ls -l

Y borrarlo con:

rm miarchivo

Piense: ¿A dónde apunta el enlace ahora?

ls -l
cat mienlace

¿Qué conclusiones obtiene?

Piense: ¿Qué ocurrirá con un enlace no simbólico (duro)?

```
# rm    mienlace
# touch         miarchivo
# ln    miarchivo       mienlace
# ls    -l      --color
# cat   >miarchivo
es este miarchivo o mienlace?
^D
#
```

¿Qué conclusiones obtiene?

Como la estructura de archivos/directorios es la de un "árbol", podemos cambiarnos al nivel precedente que se denomina "directorio padre" y se simboliza con dos puntos seguidos ".."[9], entonces con:

```
# cd    ..
```

Nos cambiamos al directorio padre, luego con:

```
# rmdir         nuevo
```

Borraríamos el directorio que habíamos creado (**¿Qué ocurre?** ¿Cómo lo soluciona?).

¿Y si intentáramos desmontar el diskette mientras estamos cambiados sobre su punto de montaje?

```
# umount        /mnt/floppy
```

Cámbiese a su "casa" e inténtelo nuevamente:

```
# cd
# umount        /mnt/floppy
```

¿Qué conclusiones saca?

Puede saber cuál es el uso del espacio en las distintas particiones/volúmenes con el comando ***disk free***:

```
# df    -h
```

La opción "-h" muestra los valores en unidades significativas para los "humanos", es decir, en kilobytes, megabytes, etc. **Observe** cuidadosamente cuánto espacio se ocupa en los distintos tipos de sistema de archivos, por ejemplo puede crear en el diskette distintos

[9] De manera análoga el directorio actual se simboliza con un punto "."

"formatos", montarlo, observar el espacio, desmontarlo, crear otro tipo de "formato", montarlo, comparar, etc.

Observe que en el CD ADIOS vienen varios tipos de sistema de archivos. Coloque el comando "mkfs" y apriete el tabulador dos veces para que el intérprete "bash" le muestre cuáles son los comandos que empiezan de esa manera.

Experimente creando un sistema de archivos tipo "ext2" en el diskette y en vez de montarlo, coloque este comando:

```
# dumpe2fs   /dev/fd0
```

Que hace un vuelco de la información del sistema de archivos.

Permisos

Como se trata de un sistema multiusuario, los archivos y directorios tienen modos o permisos de acceso, para el propietario, para el grupo al que pertenece el archivo o directorio, y para los restantes grupos (el "mundo"). Estos permisos básicamente son de lectura (**r**: read), escritura (**w**: write) y ejecución (**x**: execute). Como el permiso de ejecución carece de sentido en un directorio, la interpretación del mismo es "cambiarse a". Es decir, si un directorio no tiene el permiso "x" no podemos colocar el comando "cd" para cambiarnos a ese directorio.

Para cambiar el modo de acceso utilizamos el comando *chmod (change mode*), cuyo forma genérica sería:

```
chmod        {u, g, o, a}{+, -, =}{r, w, x}        archivo o directorio
```

Por ejemplo, si queremos habilitar el permiso de lectura al archivo "miarchivo" por parte del grupo, colocaríamos el comando:

```
$ chmod       g+r      miarchivo
```

Note que al cambiar los permisos, éstos tienen una posición fija, que está habilitado o inhabilitado. **¿Qué le recuerda esto?** ¡Acertó! Son bits. Por lo tanto, el patrón de bits "rwxrw-r—" equivale a "765" (rwx=7, rw-=6, r--=5). **Pruebe** lo siguiente:

```
$ chmod       777      miarchivo
$ chmod       000      miarchivo
$ chmod       111      miarchivo
$ chmod       222      miarchivo
$ chmod       444      miarchivo
```

Autoevaluación

1) ¿Qué método de asignación de archivos utiliza Unix?
   a) Asignación contigua.
   b) Asignación encadenada con preasignación de espacio.
   c) Indexada con asignación dinámica de espacio.
   d) Indexada con preasignación de espacio.
2) Cuando un proceso ejecuta un fork() y luego el proceso hijo un exec(), ambos procesos comparten:
   a) Nada, porque son dos procesos distintos.
   b) Los segmentos de memoria que contienen datos y pila.
   c) Los descriptores de archivos.
   d) El segmento de memoria que contiene los datos.
3) El archivo "pepe" es un enlace simbólico al archivo "juan". Ambos archivos
   a) Comparten el mismo nodo-i, que tiene el contador de enlaces igual a 2.
   b) Comparten el mismo nodo-i, que tiene el contador de enlaces igual a 1.
   c) Tienen distintos nodos-i.
   d) Tienen distintas entradas en la tabla filp, pero comparten el mismo nodo-i.
4) ¿Dónde guarda Unix la posición siguiente a leer o escribir (puntero) de un archivo por parte de un proceso?
   a) En el array de descriptores de archivos del descriptor de proceso.
   b) En la tabla FLAP.
   c) En la tabla FILP.
   d) En la tabla de nodos-i abiertos.
5) En un sistema de archivos Unix con un tamaño de nodo-i de 128 bytes, un tamaño de bloque de 1024 bytes y donde la zona de nodos-i ocupa 1024 bloques, ¿cuántos bloques ocupa el mapa de nodos-i libres?
   a) 2 bloques.
   b) 1 bloque.
   c) 4 bloques.
   d) 8 bloques.
6) En un proceso que realiza la escritura de un archivo byte a byte de forma secuencial, ¿cuál de los siguientes aspectos relacionados con una cache de bloques tiene una mayor repercusión positiva sobre el rendimiento del proceso?
   a) La cache no mejora el rendimiento para este tipo de accesos.
   b) La escritura inmediata.
   c) La escritura diferida.
   d) La lectura adelantada.
7) Un usuario con uid=12 y gid=1 es el dueño de un archivo con modo de protección "rwxr-x---". Otro usuario con uid=3 y gid=1 quiere ejecutar el archivo. ¿Puede hacerlo?
   a) Siempre.
   b) Nunca.
   c) Sólo si se le pone el bit setuid.
   d) Sólo si se le pone el bit getuid.
8) ¿Cuál de las siguientes afirmaciones es cierta?

a) Los directorios son archivos especiales orientados a carácter.
b) Un archivo especial orientado a bloque modela un dispositivo de acceso directo.
c) En MS-DOS existen los archivos especiales.
d) Un directorio es un archivo normal.

9) ¿Cuál de las siguientes secuencias de llamadas al sistema es la que correctamente redirige la entrada estándar a un archivo?
   a) fd=open(archivo, O_RDONLY); close(1); dup(1);
   b) close(0); open(archivo, O_RDONLY)
   c) close(0); fd=open(archivo, O_RDONLY); dup(fd); close(fd);
   d) fd=open(archivo, O_RDONLY); close(0); dup(0); close(fd);
10) En un sistema de archivos tipo Unix, ¿cuál es el valor del contador de enlaces asociado a un directorio que contiene un archivo y un directorio, y que está referenciado por un enlace simbólico?
   a) 4.
   b) 3.
   c) 5.
   d) 2.
11) ¿Cuál de las siguientes sentencias sobre las estrategias de asignación de espacio (organización física) a los archivos es cierta?
   a) La asignación contigua proporciona un acceso directo más rápido que la indexada.
   b) Las asignación encadenada gasta más espacio adicional para almacenar un archivo que la indexada.
   c) La asignación encadenada presenta fragmentación externa.
   d) La lectura secuencial de un archivo completo es más rápida con la asignación indexada que con la encadenada.
12) ¿Cuántos nodos-i estarán ocupados en un sistema de archivos Unix que contiene únicamente los siguientes archivos: "/f1", "/f2" (enlace simbólico a "/f1"), "/f3" (enlace no simbólico a "/f1"), y "/dir" que es un directorio vacío?
   a) 5
   b) 4
   c) 3
   d) 6
13) Un sistema de archivos tipo Unix tiene un tamaño de bloque de 2 KB y nodos-i con 12 direcciones directas, una indirecta simple, una indirecta doble, y una indirecta triple. Además utiliza direcciones de bloques de 4 bytes. ¿Qué bloques son necesarios para representar un archivo de 2 MB?
   a) Se utiliza un bloque de indexación simple y un bloque de indexación doble.
   b) Se utiliza un bloque de indexación simple y dos de indexación doble.
   c) Se utilizan dos bloques de indexación simple y uno de indexación doble.
   d) Se utilizan dos bloques de indexación simple.
14) ¿Qué es cierto con relación a la cache del sistema de archivos Unix?
   a) Los bloques de nodos-i no se mantienen en cache.
   b) Los bloques de directorio no se mantienen en cache.
   c) Acelera las lecturas pero no las escrituras.
   d) Aumenta las prestaciones y disminuye la fiabilidad del sistema.