

# MÓDULO: 7

## SISTEMA DE GESTIÓN DE ARCHIVOS

**CONTENIDO:**

- Conceptos sobre el Sistema de Gestión de Archivos.
- Administración del almacenamiento secundario.
- Métodos de acceso
- Mecanismos de integridad y seguridad
- El File System del S.O. UNIX.

**OBJETIVO DEL MÓDULO:** Describir las distintas formas de administrar el espacio de almacenamiento secundario, los métodos de acceso y los mecanismos para almacenar, modificar, recuperar y compartir la información almacenada en los soportes en forma segura como también hacer referencia a los archivos mediante un nombre simbólico.

**OBJETIVOS DEL APRENDIZAJE:** Después de estudiar este módulo, el alumno deberá estar en condiciones de:

- Explicar y reconocer las distintas estructuras de almacenamientos secundarios.
- Conocer la jerarquías de datos almacenados
- Conocer las funciones del Sistema de Gestión de Archivos.
- Comprender y explicar la organización y los métodos de acceso.
- Conocer y explicar la terminología específica empleada en éste módulo.
- Entender los mecanismos de protección utilizados en los accesos a los archivos.
- Entender los componentes del sistema de archivos que dispone el S.O. UNIX.

**Metas:**

A lo largo de este Módulo, se detallarán los conceptos básicos sobre los file systems en cuanto a describir las distintas formas de administrar los archivos en el almacenamiento secundario, el manejo del espacio de almacenamiento, estudiar las distintas funciones, operaciones sobre archivos y directorios que tienen los file systems y, también se hará una comparación entre los file systems más usados en la actualidad. Esta comparación se basará sobre los aspectos que tienen los cuatro sistemas de archivos más importantes en cuanto a su filosofía, la estructuración, funcionamiento, seguridad y protección. En particular daremos una amplia descripción del Sistema de Archivos de UNIX.

## 7. Sistema de Gestión de Archivos (File System)

### 7.0. Introducción

Para hacer un uso conveniente del espacio de almacenamiento, el sistema operativo hace una abstracción de las características físicas de los dispositivos de almacenamiento para definir una unidad lógica de almacenamiento llamada *archivo* (*file*).

El módulo del sistema operativo encargado de la tarea de administrar el espacio de almacenamiento en un dispositivo y las estructuras abstractas de datos, lo denominamos *Sistema de Gestión de Archivos* o *File System (FS)* o simplemente *sistema de archivo*.

Las computadoras pueden almacenar información en diferentes medios de almacenamiento, como ser discos magnéticos, cintas o discos ópticos. El sistema operativo nos da una visión lógica acerca del almacenamiento de la información, abstrayéndose de las propiedades del dispositivo físico de almacenamiento, y define al *archivo* como la unidad lógica de almacenamiento.

En este módulo comenzamos este tema con una breve discusión de objetivos, atributos, conflictos, estructura y función de un Sistema de Gestión de Archivos. Luego, se explicamos las distintas operaciones que se pueden hacer sobre un Archivo. Inmediatamente trataremos el espacio de almacenamiento tanto libre como el asignado. También estudiaremos las estructuras de directorios y los métodos de accesos. Finalmente se ejemplifica el Sistema de Archivos utilizado por UNÍS a través de estudio de casos

### 7.1.- Concepto de Sistema de Gestión de Archivos

Es el conjunto del software del sistema que provee tanto a usuarios como a aplicaciones servicios en el uso de archivos. La única forma en que un usuario o una aplicación puede acceder a archivos es a través del sistema de gestión de archivos. Esto primero libera al usuario (o programador) de la necesidad de desarrollar un soft de propósito especial para cada aplicación y segundo le proporciona pertenencias al Sistema operativo con un medio (herramienta para controlar sus bienes (objetos) mas importantes.

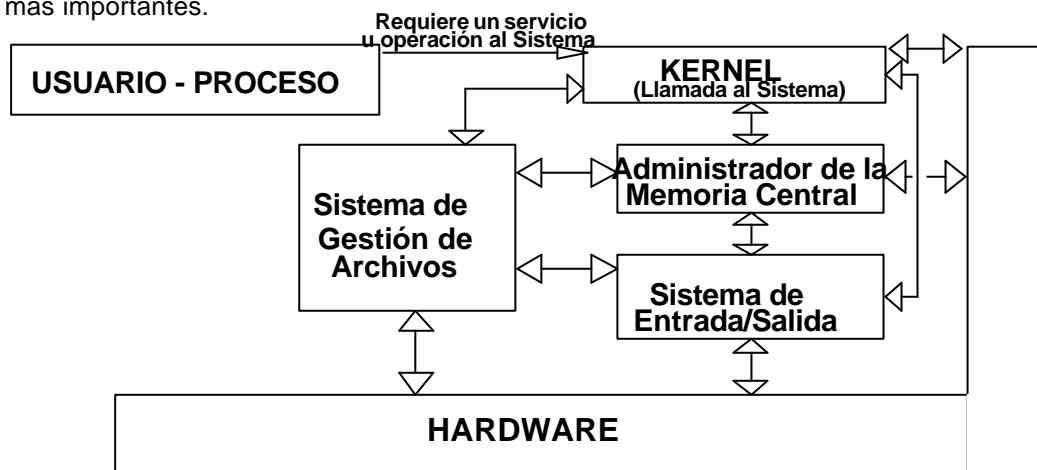


Fig. 7.01 Interacción del Sist. de Gestión de Archivos con otros módulos del S.O.

El Sistema de Gestión de Archivos (lo llamaremos indistintamente Sistema de Archivos - File System o Sistema de Administración de Archivos) debe ocultar a los usuarios la complejidad de la manipulación de la estructura abstracta de datos, y su almacenamiento, y proporcionarle una visión simplificada y uniforme de un nombre desde el punto de vista lógico.

Para cumplir con estas consideraciones necesita:

- Traducir un pedido de acceso desde el espacio lógico de direcciones del archivo a un espacio físico en el soporte.
- Transferir los datos entre el Archivo y la Memoria Central o viceversa.
- Llevar la gestión del espacio de almacenamiento en forma automática y transparente
- Brindar un soporte para proteger, compartir y recuperar los datos de un archivo y también facilitar la restauración del archivo ante operaciones equivocadas o fallas del Sistema. El

soporte de información puede ser desde una memoria basada en semiconductores, cintas magnéticas, discos magnéticos, ópticos o magneto-ópticos. El soporte puede contener uno o más archivos o el archivo, si es muy voluminoso, necesitar más de un soporte (multi-volumenes).

Se dice que el soporte está **montado (mount)** cuando está en línea (controlado por el procesamiento). El soporte puede ser desmontado (si es removible del periférico) y ser almacenado físicamente en un lugar apropiado (Cintoteca o Discoteca) o simplemente dejado fuera de línea (off line).

En este texto solo estudiaremos los archivos que están en línea y en particular, para simplificar, trataremos los discos para describir los conceptos sobre el Sistema de Gestión de Archivos y sus servicios.

El Sistema de Gestión de Archivos se integra con otros Administradores en las distintas capas o niveles que presentamos al S.O.. La integración se realiza entre el Kernel, el Administrador de Memoria Central y el Gestor del Sistema de Entrada / Salida.

Si los archivos son almacenados en un disco, el sistema operativo divide a los mismos en *bloques* de longitud fija (típicamente de 512 a 2048 bytes) para su almacenamiento. Mientras que en una cinta los bloques pueden tener cualquier longitud.

### 7.1.1.- Objetivos de un Sistema de Gestión de Archivos

- 1 Satisfacer los requerimientos y las necesidades del usuario en cuanto a administración de datos, que incluye: a) el almacenamiento de los datos y b) poder efectuar las operaciones anteriormente mencionadas (operar con archivos, registros y campos).
- 2 Garantizar (en la medida de lo posible) la validez de los datos almacenados en cada archivo.
- 3 Optimizar el desempeño: a) desde el punto de vista del sistema: overall throughput (rendimiento de todos los componentes) y b) desde el punto de vista del usuario: en términos de tiempos de respuesta cortos. Como se ve, son objetivos contrapuestos.
- 4 Proveer soporte de E/S para una amplia variedad de dispositivos de almacenamiento.
- 5 Minimizar (o eliminar) la posibilidad de pérdidas o destrucción de datos.
- 6 Proveer un conjunto standard de rutinas de interfases de E/S.
- 7 Proveer soporte de E/S para usuarios múltiples, en el caso de sistemas multiusuarios.

En cuanto al primer objetivo los requerimientos dependerán de la variedad de aplicaciones o ámbitos en el cual se use el sistema de computación.

### 7.1.2.- Requerimientos mínimos para un sistema interactivo de propósito general

Para un sistema interactivo de propósito general los REQUERIMIENTOS mínimos son:

Cada usuario debe poder:

- 1- crear, borrar, leer y modificar archivos.
- 2- tener acceso controlado a archivos de otros usuarios.
- 3- controlar que tipo de acceso permite sobre sus archivos.
- 4- reestructurar sus archivos en forma apropiada a cada problema.
- 5- mover datos entre archivos.
- 6- efectuar copias de resguardo y restaurar sus archivos en caso de daño.
- 7- acceder a sus archivos por medio de NOMBRES SIMBÓLICOS.

Tanto los **objetivos** como los **requerimientos** deberán ser tenidos en cuenta durante toda nuestra discusión acerca del Sistema de administración o gestión de archivos.

### 7.1.3. Sistemas basados en Cinta.

Los primeros sistemas de archivo fueron soportados en cinta. La ventaja de esta solución es su simplicidad, pero sufre de una cierta ineficiencia, ya que las cintas son bastante largas (por ejemplo 727 metros -2.400 pies-). Se ha demostrado que la mayoría de los archivos generados por aplicaciones son relativamente pequeños y al ser de poco tamaño solo requiere unos pocos centímetros o metros de cinta, pudiendo una cinta almacenar varios Archivos y los accesos se dificulta por ser secuencial.

Otro problema se presenta con archivos muy grandes, donde se necesitan varias cintas. La mayoría de los sistemas soportaban archivos en cinta multi-volumenes o multi-reel. Para resolver el problema de los archivos chicos, se crearon sistemas que podían almacenar múltiples archivos en una cinta, aumentando la utilización de las cintas. Surge, sin embargo, el problema de saber que archivos residen en una cinta. Para esto se agrega un *directorio a la cinta*. En IBM, una cinta es llamada volumen y el directorio *Volumen Table Of Content -VTOC-* (*Tabla de Contenidos del Volumen*). El directorio lista el nombre y locación de cada archivo en cinta. Este concepto también se usa en los

Discos.

Cada cinta, o dispositivo, tendrá su propio directorio: este es un lugar conveniente para guardar información resumida de los archivos almacenados en ese dispositivo.

Encontrar un archivo implica identificar la cinta apropiada, buscar en su directorio y posicionar la cinta en el archivo adecuado. La única diferencia ahora es que el fin-de-archivo (EOF End of File) no coincide con el fin-de-cinta (EOT End Of Tape). Las separaciones las efectúa la Unidad de cinta colocando unas marcas ( Tape Marks T.M.)

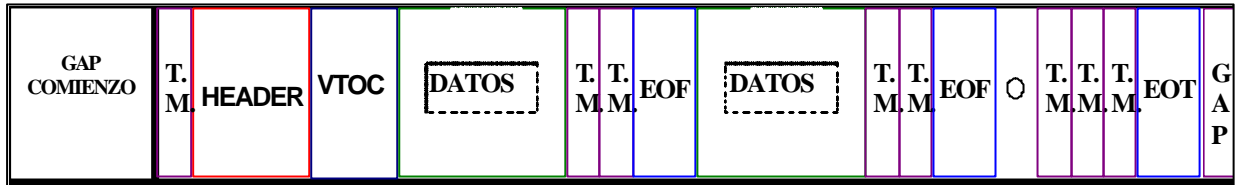


Fig. 7.02 organización de una cinta

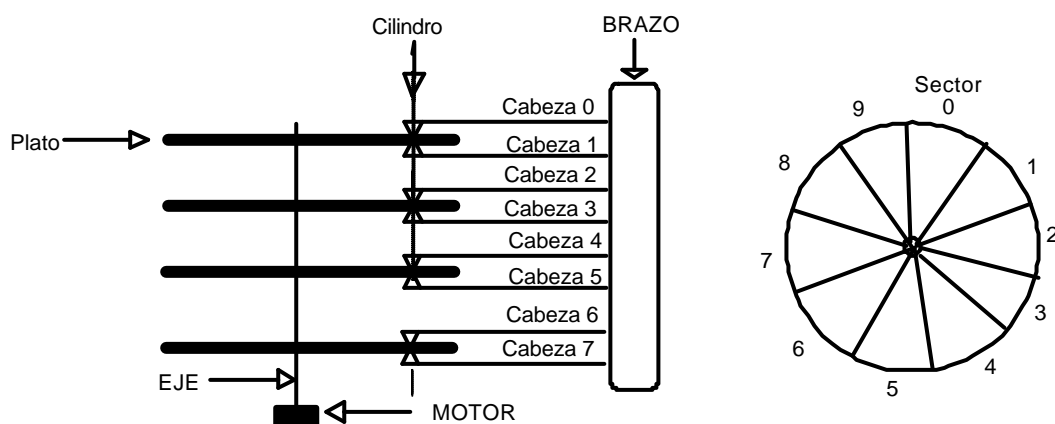
También existen otras dificultades, por ejemplo el excesivo consumo de tiempo cuando un programa lee alternadamente líneas desde 2 archivos en la misma cinta, o efectúa una copia de un archivo en otra cinta para ser leída en otra unidad. Esto obliga esperar bastante tiempo por el rebobinado y el avance de la cinta. Otro problema que se presenta es para re-escribir la información de un archivo, para ello se debe procesar todos los datos que contiene el resto de la cinta en Memoria Central y luego Grabar nuevamente la Cinta y modificar el directorio que usualmente está al comienzo de la cinta con el consiguiente rebobinado y retardo en el tiempo.

#### 7.1.4. Sistemas basados en Disco.

Muchos de los problemas asociados con el almacenamiento en cinta, se resuelven con sistemas basados en disco. Un disco se divide en pistas, cuyo número varía de dispositivo a dispositivo. Cada pista se divide en sectores. El sector es la más pequeña unidad de información que puede ser leída o escrita en un disco.

Dependiendo de la unidad, los sectores varían de 512 bytes a 1024; de 9 a 65 sectores por pista y desde 75 a 3000 pistas por cada superficie. Los sistemas más grandes pueden tener varios platos de discos (cada plato con 2 superficies: superior e inferior). Para acceder a un sector se debe especificar la pista (cilindro), la superficie o cara (Cabeza Lectora/ Escritora) y el sector. Estos elementos constituyen la *dirección en el soporte* (Cylinder, Head, Sector). En la **figura 7.03** se ejemplifica el concepto de cilindro, cara y sector.

Las cabezas se mueven a un lugar determinado (tiempo de búsqueda o seek time) y a la superficie correcta; luego debe esperar para que el sector pedido llegue a pasar debajo de la cabeza (latencia rotacional). Un *cilindro* es un conjunto de pistas que están en la misma posición en el disco, pero en diferentes platos. No se necesita ninguna búsqueda "seek" para acceder a pistas del mismo cilindro.



OBSERVACIÓN: En este ejemplo cada pista tiene 10 Sectores (0..9) y hay 8 pistas por Cilindro (Cabezas 0 .. 7) lo que implica que se procesan 80 sectores sin mover el brazo con las cabezas.

FIGURA 7.03 Elementos de un disco rígido magnético

La transferencia de E/S entre memoria y disco se realizan en unidades de uno o más sectores. El direccionamiento de un sector en particular requiere un número de pista, de superficie y de sector, por lo que podemos considerar como un arreglo tridimensional de sectores en un disco. Comúnmente el

S.O. trata al disco como a un arreglo unidimensional de *bloques de disco*, donde cada bloque contiene uno más sectores. Típicamente las direcciones de un bloque se incrementan a partir de todos los sectores de un cilindro, luego todas las pistas de un cilindro y finalmente desde cilindro 0 al último del disco. Si *ns* es el número de sectores por pista y *np* el número de pistas por cilindro, entonces podemos convertir una dirección de disco dado por cilindro *c*, superficie *h*, sector *s* a un número de bloque unidimensional *b*:

$$b = s + ns * (h + c * np)$$

Observemos que con esta forma de mapeo, el acceso al bloque *b+1*, cuando *b* fue el último accedido y es el último bloque de un cilindro y *b+1* el primero del próximo, requiere un movimiento de cabezas de una sola pista.

La diferencia fundamental con cinta, es el acceso directo a los bloques. También los discos requieren de un directorio de dispositivo indicando que archivos residen en el mismo. Se puede re-escribir un bloque, modificarlo, leerlo y re-escribir el directorio sin necesidad de tener que copiar el resto del disco.

### Bloqueo en Discos:

Los discos tienen un tamaño en bloques perfectamente definido por el tamaño de los sectores. Todas las operaciones de Entrada/Salida manipulan datos contenidos en *bloques (registros físicos)* y todos los bloques son de igual tamaño. Las cintas son más flexibles ya que el tamaño del bloque físico puede definirse por software. Por ejemplo, históricamente era muy común una longitud de 80 caracteres (imagen tarjeta) por registro físico. Esto producía bastante desperdicio debido a los huecos entre registros (Inter Record Gaps - IRG). De todos modos, tanto en cinta como en disco, es probable que el tamaño del registro físico no coincida con el registro lógico. Aun más, los registros lógicos pueden llegar a ser de longitud variable. La operación de agrupar un número de registros lógicos en bloques físicos se conoce como *empaquetado*.

El empaquetado puede realizarse desde una aplicación por el usuario o por el S.O. En ambos casos, el archivo puede considerarse como una secuencia de bloques. Todas las funciones básicas de E/S operan en términos de bloques.

Observemos que siempre asignar espacio en disco mediante bloques implica que una porción del último bloque se desperdicia. Si tenemos bloques de 512 bytes, entonces un archivo de 1949 bytes tendría 4 bloques (2048 bytes), donde los últimos 99 bytes se pierden. Esto se conoce como *fragmentación interna* del bloque. Todos los Sistemas de Gestión de Archivos sufre el fenómeno de la fragmentación. En mayor medida cuando mayor sea el tamaño del bloque.

### Ablocamiento de registros (Record blocking).

**Registros (Records):** son la unidad lógica de acceso a un archivo.

**Bloques (Blocks):** son la unidad de E/S (de almacenamiento).

En la mayoría de los sistemas los bloques son de longitud fija. Esto simplifica la entrada y salida, asignación de buffer en memoria central y la organización de los bloques en el almacenamiento secundario.

Problema: ¿Cuál debería ser la longitud relativa de un bloque cruzado con la longitud promedio de los registros? Cuanto mayores sean la longitud del bloque mayor la cantidad de registros que se intercambian en una sola operación de entrada salida. Si el archivo se procesa o se busca (Search secuencialmente), esto es ventajoso, y disminuye el numero de operaciones E/S y por lo tanto decrece el tiempo de procesamiento. Si los registros se acceden al azar implica que se está aumentando el tiempo de transferencia en cada operación de entrada salida (se está transfiriendo registros que no se usan). Combinando la frecuencia de operaciones secuenciales con la potencial vecinidad (locality) de referencias se puede decir que el tiempo de transferencia de memoria central a memoria secundaria se reduce usando bloques mas largos. Desventaja: Bloques mas largos requieren mayores buffers E/S, dificultando la administración de los buffer.

Dada una longitud de bloques igualdad hay 3 métodos de ablocamiento posibles:

- 1) **Ablocado fijo (fixed blocking):** Se usan registros de longitud fija. En cada bloque se almacenan un número fijo de registros. Aquí hay fragmentación interna (a cada bloque). Espacio sin usar al final de cada bloque.
- 2) **Ablocado de registros de longitud variable, registros expandidos (desmembrados: spanned records):** se empaquetan registros de longitud variable sin espacio inutilizable (libre). De esta forma un registro se puede expandir en dos bloques fijos. La continuación se indica apuntando al bloque siguiente.

- 3) **Ablocado de registros de longitud variable, registros no expandidos:** se usan registros de longitud variable pero no se usa expansión. Como un registro lógico no se puede expandir sobre dos o mas bloques físicos, esto implica que hay espacio sin usar (libre) al final de cada bloque. Ablocado fijo es el modo común para archivos secuenciales con registro de longitud fija. Los registros expandidos permiten un almacenamiento eficiente y no limita la longitud de los registros. Los registros que se expanden son dos bloques requieren dos operaciones de entrada salida por cada acceso. Es una técnica de difícil implementación. Los archivos son difíciles de actualizar independientemente de su organización. Los registros de longitud variable sobre registros no expandidos generan un malgasto de espacio (espacio sin usar “al fondo” de cada bloque que se conoce como Fragmentación interna) y limita la longitud máxima de un registro, a la longitud de un bloque. La técnica de ablocar de registros puede interactuar con el hardware de memoria virtual. En un ambiente de memoria virtual es deseable hacer que la página sea la unidad básica de transferencia (Bloque). Si la página es muy chica no es práctico tratar a una pagina como un bloque para bloqueo no expandido. Algunos sistemas combinar varias páginas para crear un bloque más largo, con el propósito de mejorar la E/S del archivo. Ej.: archivos VSAM de IBM.

### 7.1.5. Objetivos y Funciones del Sistema de Gestión de Archivos.

El principal objetivo de un Sistema de archivos es permitir las operaciones y accesos en forma segura sobre los soportes para almacenar, modificar, eliminar o recuperar la información generada por los programas o aplicaciones y administrar el espacio de almacenamiento secundario.

Podemos definir las siguientes funciones de un Sistema de Gestión de Archivos:

- Permitir la creación, modificación y borrado de archivos sea por programas, usuarios o el propio S.O..
- Permitir el acceso por los usuarios a los archivos, el uso compartido cuidadosamente controlado de los mismos y evitar el acceso no autorizado por parte de terceros. Estos controles involucra los accesos a lecturas (privacidad), a escritura y a ejecución, o combinaciones (de algunos o los tres).
- Realizar en forma automática la gestión de la memoria secundaria.
- Permitir hacer referencias a un archivo mediante un nombre simbólico.
- Proteger a los archivos frente a fallas del sistema o contra accesos no autorizados o contra movimiento de los datos.
- Mantener el almacenamiento en línea.
- Suministrar los medios de organización y accesos a los datos en forma conveniente para los usuarios o las aplicaciones que lo requieran.
- Mover los datos entre el archivo y Memoria Central en colaboración del administrador de Entrada / Salida.

### 7.1.6.- La Gestión del File System

Otra forma de ver la funcionalidad del file system es siguiendo el diagrama de izquierda a derecha. Los usuarios y los programas de aplicación interactúan con el file system por medio de comandos para crear y borrar archivos y para efectuar distintas operaciones sobre los registros de los archivos.

Antes de realizar cualquier operación, el file system debe identificar y localizar el archivo seleccionado. Para ello necesitará de algún tipo de directorio (índice-catálogo) que describa la ubicación (física) de todos los archivos, mas sus atributos. Si es un sistema multiusuario se debe controlar que solo accedan a ese archivo los usuarios autorizados y para realizar las acciones permitidas (R,W,X,Tx).

En general las operaciones básicas a efectuar sobre los registros de un archivo dependen de su estructura (organización de los registros) y de la forma de accederlos. Para traducir los comandos de usuario a comandos específicos de manejo del archivo, se debe emplear el método de acceso adecuado a la estructura del archivo.

Mientras que usuarios y programas ven al archivo como un conjunto de registros, la E/S se efectúa a nivel bloques. Por lo tanto los registros de un archivo deben ablocarse para su Salida y desblocarse para su Entrada. Para llevar a cabo estas operaciones de E/S se necesitan de varias funciones. Se debe manejar el almacenamiento secundario, es decir, asignar bloques de almacenamiento secundario a medida que el archivo vaya creciendo, y liberando el espacio que el archivo deje de usar; todo esto se hace en base a bloques (manejo del espacio libre y del espacio ocupado sobre cada soporte de almacenamiento de información). Además deben planificarse los

pedidos de E/S sobre bloques en particular. Las funciones de planificación de disco y de asignación de archivos hacen a la performance del dispositivo, y se deben analizar en forma conjunta. Esta optimización dependerá de dos factores: de la estructura (organización) del archivo y del método de acceso.

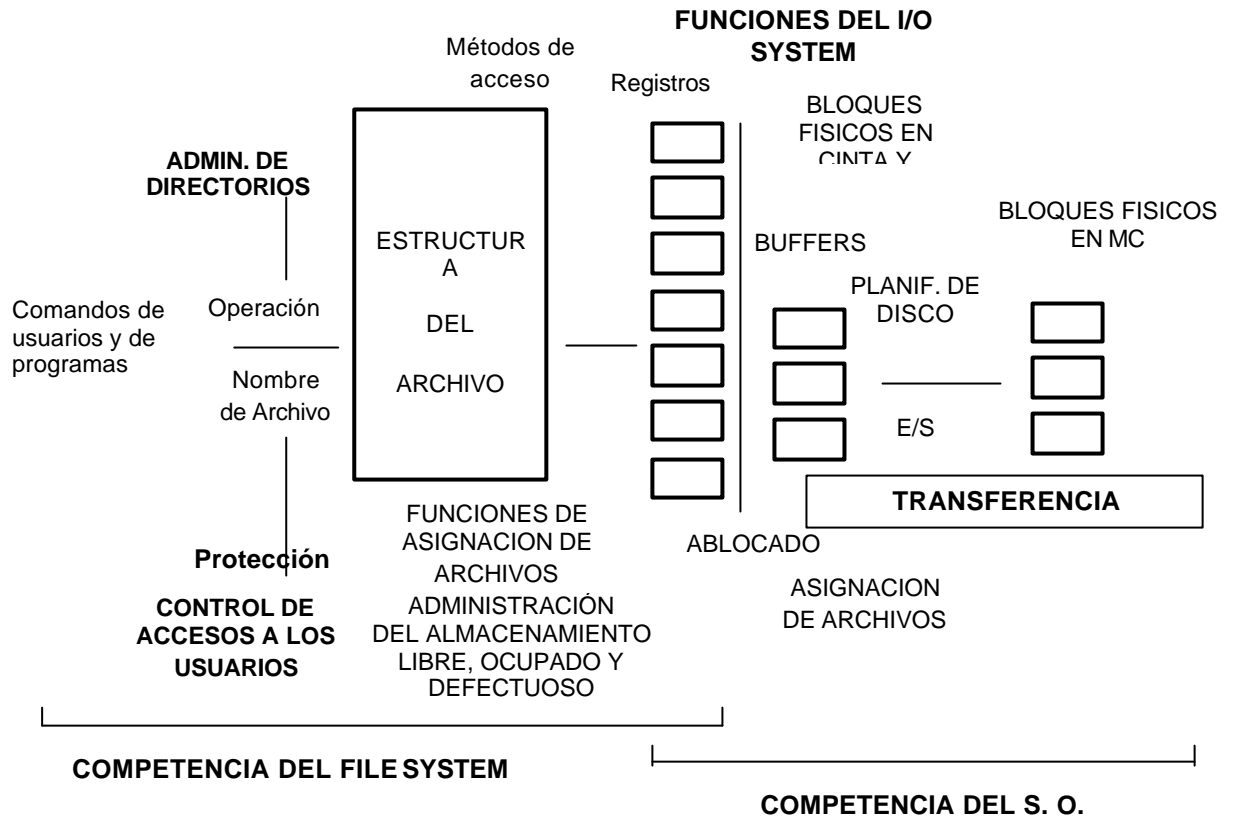


Figura 7.04 Competencias entre File System y el S.O.

Desarrollar un sistema de gestión de archivos óptimo (desde el punto de vista del rendimiento (desempeño)) es una tarea complicada al máximo. El gráfico precedente sugiere una división entre los que se consideran aspectos inherentes al sistema de gestión de archivos como un utilitario del sistema, y los aspectos inherentes al sistema operativo. Con un punto de intersección que es el procesamiento de registros.

Los aspectos a ser tenidos en cuenta al diseñar un file system se van a basar en las **organizaciones de archivo** y los **métodos de acceso** soportados por el mismo.

Luego nos adentraremos en el concepto de directorio (índice) de archivos, que a veces es administrado por el sistema operativo como un servicio al file system.

Otros tópicos tratarán con aspectos físicos de la E/S relacionados con la administración de archivos, y esos serán aspectos propios del diseño del sistema operativo.

Uno de esos asuntos es la forma en que los registros lógicos se organizan en bloques físicos (ablocamiento de registros). El File System tratará las cuestiones asociadas con la asignación de espacio (en el almacenamiento secundario) a cada archivo, y con la administración del espacio libre en el almacenamiento secundario.

### 7.1.7. Conflictos

Antes de caracterizar un Sistema de Gestión de Archivos debemos plantear algunos conflictos que se presentan cuando se desea administrar la memoria secundaria. Es fundamental conocer estos conflictos, porque a partir del planteo de las distintas soluciones a los mismos se construyen los módulos del Sistema de Gestión de Archivos:

- Determinación de espacio libre, ocupado y defectuoso.
- Determinación del espacio ocupado por un archivo.
- Determinación de la dirección en el soporte de un archivo.
- Organización del directorio y de los archivos.
- Los Métodos de Accesos.
- Protección.

## 7.2. ALMACENAMIENTO SECUNDARIO

El S.O. es el encargado de asignarle bloques a los archivos, y es aquí donde surgen 2 cuestiones administrativas:

- 1- Se debe asignar espacio de almacenamiento a los archivos
- 2- Se debe tener bajo control el espacio disponible para efectuar las asignaciones.

Estas 2 tareas están íntimamente ligadas: el esquema utilizado para asignar espacio a los archivos va a influir en el esquema utilizado para **controlar el espacio libre**.

Existe además una última relación entre las estructuras de archivos y las políticas de asignación de bloques a los archivos (file allocations).

### 7.2.1 Alternativas para la Asignación de espacio a los archivos (File Allocation) en un solo disco

Para el tratamiento de este tema se deben tener en cuenta varias cuestiones:

- a) ¿Se debe asignar a un archivo la longitud máxima posible en el momento de crearlo?
- b) El espacio a asignarle a un archivo ¿constará de una o mas unidades (sectores o bloques) contiguas (llamadas porciones)? El tamaño de las porciones van desde un sector hasta una única porción contigua para todo el archivo. ¿Qué longitud se va a usar? ¿Qué tamaño tendrá cada porción?
- c) ¿Qué estructura de datos (tablas, archivos, bloques, etc.) se va a usar para controlar todas las porciones asignadas a cada archivo? A esta tabla se la suele llamar FAT (File Allocation Table).

Examinemos estos 3 aspectos, uno por vez:

- a) **Preasignación vs. Asignación Dinámica:** Para preasignación se necesita conocer de antemano cuál será la longitud máxima a ocupar por el archivo. En el caso de compilaciones, archivos resumen de datos, transferencia de archivos, etc., no es difícil conocer de antemano este dato. Pero para muchas aplicaciones el tema no es tan sencillo, en cuyo caso el analista estima muy groseramente el posible tamaño máximo, otorgándole al archivo una medida superior (con el consecuente desaprovechamiento de espacio secundario) para que los procesos nunca se suspendan o aborten por falta de espacio. La ventaja de la asignación dinámica radica en que el espacio se va otorgando a medida que el archivo va creciendo, o sea, a medida que necesita se le va otorgando una porción más (que se agregará a la FAT del archivo y se quitará del registro de espacios disponibles - libre).
- b) **Longitud del tamaño de la Porción (a asignar):** Puede ser una sola porción a todo el archivo o un sector por cada vez que el archivo lo pida. Aquí hay un compromiso entre eficiencia (desde el punto de vista de uso del archivo) y eficiencia total del sistema. En este compromiso se deben tener presentes al menos 4 aspectos:

- 1) La contigüidad del espacio asignado aumenta el rendimiento "bastante" si se tiene un S.O. orientado a transacciones y "muchísimo" si al archivo se lo accede totalmente y secuencialmente.
- 2) Si se asigna porciones pequeñas cada vez, entonces se estará aumentando notablemente la longitud de la tabla necesaria para manejar la información de asignación de espacio al archivo. Como a su vez la tabla de espacio libre.
- 3) El trabajar con porciones de longitud fija (Ej.: n bloques por vez) simplifica la reasignación de espacios libres.
- 4) El operar con porciones de longitud variable o con porciones pequeñas de longitud fija, minimiza el derroche de espacio "sobreasignado" a cada archivo.

Estos 4 aspectos interactúan entre sí por lo que se los debe considerar en forma conjunta (no aisladamente). Como resultado surgen las 2 alternativas siguientes:

- 1) **Porciones largas y contiguas, de longitud variable.** La longitud variable evita derroches, la contigüidad hace que las FAT sean pequeñas. Como desventaja: El espacio es muy difícil de reusar (reasignarse).
- 2) **Bloques pequeños de longitud fija** proveen de gran flexibilidad. Su asignación requiere de tablas grandes o estructuras complejas. Se abandona la contigüidad, los bloques se asignan cuando se necesitan.

Estas 2 opciones son compatibles con preasignación y con asignación dinámica. Para el caso 1, al archivo se le preasigna un grupo contiguo de bloques, con lo cual se elimina la FAT; basta con un puntero al primer bloque y la cantidad de bloques que conforman esta porción única. En el caso de asignación fija de bloques, se asigna en la creación del archivo, todas las porciones



necesarias y queda una FAT de longitud fija.

Con porciones de longitud variable se tiene que enfrentar el problema de la fragmentación del espacio en disco. Esto es semejante al manejo de la memoria central.

#### Estrategias Posibles:

- **FIRST FIT:** tomar de la lista de bloques libres (Free Block List) el primer grupo de bloques contiguos sin uso, de medida lo suficientemente grande como para satisfacer el pedido.
- **BEST FIT:** tomar de la Free Block List un grupo de medida lo suficientemente grande como para satisfacer el pedido.
- **NEAREST FIT:** tomar de la Free Block List un grupo de medida lo suficientemente grande que esté lo mas cercano posible a la anterior asignación (para que de esta forma se incremente la localidad (cercanía)).

No se puede determinar cual estrategia es la mejor y la dificultad en establecer un modelo de estrategias alternativas radica en la gran cantidad de factores interactuantes en el modelado:

- Tipos de Archivos (SEC, I-S, Indexados, Random, Directos).
- Patrón General de Acceso a los Archivos (SEC, Random, Dinamic).
- Grado de Multiprogramación.
- Otros Factores de rendimiento del sistema (Disk Catching, Disk Scheduling).
- Etc.

### 7.2.2.- Catalogación de la información en el soporte

Para poder asignar y desasignar espacio en el soporte el Sistema Operativo divide el disco básicamente en tres áreas:

- Área de datos fijos. (Fixed Data Area - FDA, que incluye la Tabla de Particiones y el Sector de Booteo (partition table y Boot Sector)
- Área de catálogo: área del Espacio libre y del Espacio ocupado (Free Space List y File Allocation Table). Esta última está compuesto por un área específica de Directorio (File Directory Block) de varios niveles, que incluye al Master File Directory y al User File Directory.
- Área de datos (Donde se localizan físicamente los datos de los archivos).

#### a) Área de datos fijos (FDA Fixed Data Area)

Este área contiene datos tales como el comienzo y fin de cada partición, tamaño, estado de la partición (activa o inactiva), etc., Además contiene el sector de booteo, la cantidad de bloques que dispone el disco, cantidad de entradas en el directorio, cantidad de sectores por bloque, dirección de comienzo del directorio, etc.

Es inicializada durante el formateo de alto nivel. Son datos críticos que deben ser apropiadamente protegidos por el sistema operativo.

Generalmente está localizada en los primeros sectores del disco y la llamaremos *BPB (Bios Parameter Block. Bios significa Basic Input Output System, partition table + Boot Sector)*,

#### b) Área de Catálogo

Es el área donde el sistema operativo almacena el directorio, la tabla de asignación de espacio, la lista de espacio libre del disco y la lista de sectores defectuosos (inutilizados).

En los siguientes puntos del texto veremos algunas técnicas para manejar los distintos componentes de esta área.

#### c). Área de Datos

Finalizada el área de catálogo, el Sistema Operativo asigna todo el resto del disco libre para el almacenamiento de los archivos del usuario.

## 7.3. Administración del espacio de almacenamiento

La Gestión del Almacenamiento Secundario, básicamente está representado por la asignación del espacio, cuyo objetivo es utilizar eficazmente al espacio y posibilitar el acceso rápido a la información almacenada.

Depende fundamentalmente del tipo de soporte (Disco, Cinta, etc.) y es controlado y actualizado, por el Sistema de Gestión de Archivo cada vez que se Borra, se Crea o se graba un archivo en un

dispositivo.

El Soporte puede ser particionado en partes. Entonces el mismo soporte tendrá una o más particiones que se guardan en la *Tabla de Particiones*.

El espacio de almacenamiento se relaciona entre lo que no está asignado (libre) y el que está ocupado por los archivos (asignado) y generalmente se utiliza una tabla para administrar ese espacio llamada *FAT (File Allocation Table)*.

### 7.3.1. Administración del Espacio Libre.

Todos los métodos de asignación de espacio en disco (a archivos) se basan en el uso de una tabla de asignación de disco. Las **FAT** se estructuran en base a la **DAT** (Disk Allocation Table).

Es necesario poder re-usar el espacio liberado cuando se borra un archivo en disco.

Normalmente el S.O. tiene una *lista de espacios libres* ya sea en la FAT o en algún otro lugar del soporte llamado Disk Allocation Table, en la que se guardan todos los bloques libres. Al crear y al grabar un archivo, se realiza una búsqueda en ésta tabla y se asignan bloques, los que son removidos de la lista de libres. Al borrarse un archivo los bloques se agregan a la lista. Puede ser, que la lista de espacios no se implemente como una lista sino por algún otro método como veremos a continuación, pero genéricamente la llamaremos lista.

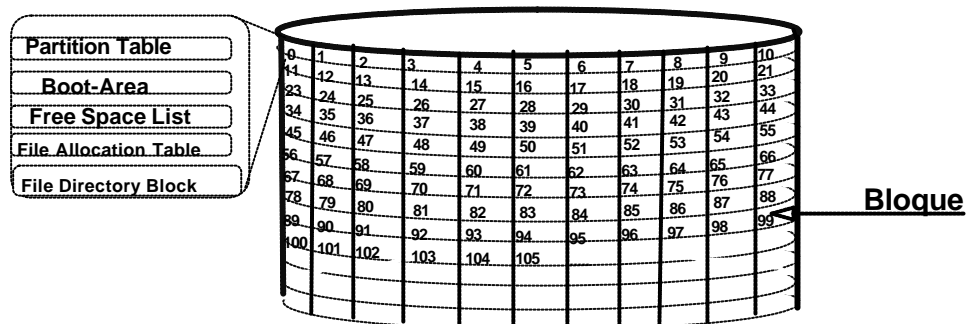


Figura 7.05 - Tablas del Soporte

#### a) Bit Map o Bit Vector (Mapa De Bits)

Frecuentemente se implementa como un vector o mapa de bits en la FAT. Cada bloque del disco se representa con un bit. Si el bloque está libre se le asigna al bit un "0", caso contrario el bit vale un "1" que indica que está ocupado.

Por ejemplo: Supongamos un disco donde los bloques 2, 3, 4, 5, 8, 9, 15 y 19 están libres; el mapa de bits del disco sería:

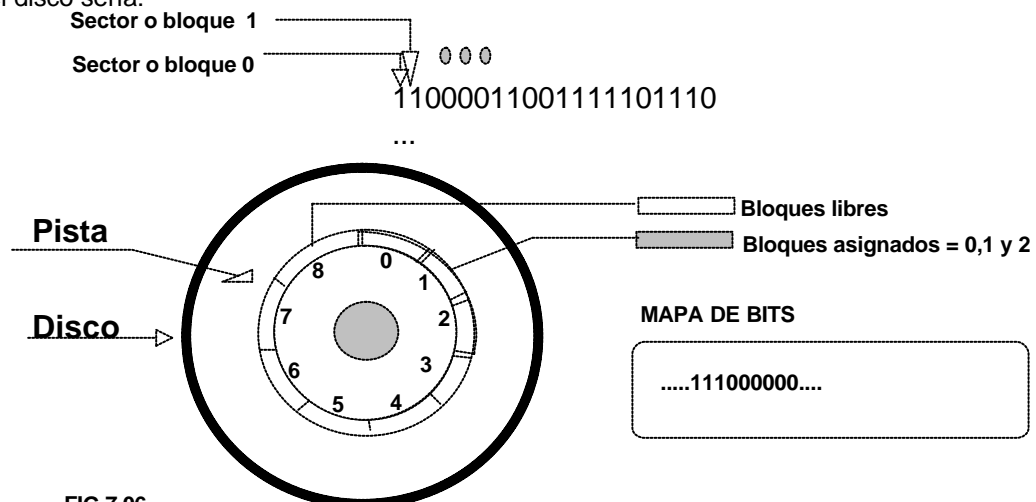


FIG 7.06

Por ej. Si el disco es de 16GB, su tabla ocuparía 4MB (demasiado). Tener la tabla totalmente en disco (8000 bloques de 512 bytes) no es operativo a disco casi lleno. En memoria debemos tener al menos "algo" de esta tabla. La mayoría de los sistemas que, como DAT, emplean la tabla de bits, mantienen en memoria central una serie de datos auxiliares de esta tabla de bits que resumen el contenido de cada uno de los subrangos en que he dividido la tabla original. Generalmente se subdivide la tabla en una serie de subrangos de igual longitud. La tabla resumen tendrá una entrada por cada subrango, y en esta entrada puedo colocar la cantidad total de sectores libres y la cantidad máxima de sectores libres contiguos que tengo disponibles en ese subrango. Cuando un archivo pide una cierta cantidad de

bloques contiguos, recorrerá la tabla resumen para ubicar el subrango apropiado. Luego se opera en memoria con ese subrango (trayéndolos del bloque de disco correspondiente) actualizándolo en memoria central y luego se lo regraba en el sector traído a memoria. Finalmente se actualiza la tabla resumen que se tiene en memoria central.

**Ventajas:**

Sencilla implementación.

Es relativamente fácil encontrar un bloque (o un grupo contiguo de bloques) libre.

La tabla es de menor tamaño posible (usa un bit por cada bloque).

Gran velocidad de búsqueda.

**Desventajas:**

No posee grandes desventajas.

**c) Lista Enlazada de Bloques Libres (Chained Free Portions):**

Otra solución es vincular (-link-) todos los bloques libres, guardando un puntero al primer bloque libre. Este bloque tiene un puntero al siguiente bloque libre y así sucesivamente. En nuestro ejemplo de la **Figura 7.07**, se guarda un puntero al bloque 1. El bloque 2 tiene un puntero al 4, este apunta al 6, etc. De esta manera los bloques libres del disco poseen un puntero al siguiente bloque libre. Lo único que tiene que guardar el sistema es el puntero al primer bloque libre.

El sistema enlaza todos los bloques libres del disco. El comienzo de la lista lo marca el puntero *Free Space List Head (FSLH)*.

Este método consume muy poco espacio ya que no necesita de una disk allocation table, solo necesita de un puntero al comienzo de la cadena y la longitud de la primera porción. Este método sirve para los tres métodos de asignación de espacios a archivos (Contigua, Encadenada e Indexada). Si la asignación se efectúa de a un solo bloque por vez, simplemente hay que tomar el primer bloque (que esta a la cabeza de la cadena) y ajustar el primer puntero y la primer longitud de porción. Si la asignación es con porciones de longitud variable se puede usar un algoritmo **FIRST FIT**. Los encabezados (headers) de las porciones se cargan en memoria (fetch) de a uno por vez para igualar la próxima porción libre disponible en la cadena. Nuevamente se debe ajustar los punteros y los valores de longitud correspondientes. Este método tiene sus problemas propios: luego de un cierto tiempo de uso el disco se fragmenta enormemente, y muchas posiciones se tornan del tamaño de un solo bloque (sector). Es de notar que cada vez que se asigna un bloque primero se debe leer ese bloque para recuperar el puntero del nuevo primer bloque libre antes de escribir datos en ese bloque. Para ser una asignación a un archivo de varios bloques a la vez, debo asignar los bloques de a uno por vez (debido a la alta fragmentación del disco). Esto demora la creación del archivo. El borrar archivos muy fragmentados consume demasiado tiempo.

**Ventajas:**

- Muy poco espacio ocupado, ya que lo único que hay que almacenar es el puntero al comienzo de la lista.

**Desventajas:**

- Es muy poco eficiente ya que para recorrer la lista, debemos leer cada bloque del disco lo cuál lleva un gran tiempo de E/S (Entrada/Salida).
- Altamente vulnerable, si se corta un enlace, el sistema pierde completamente el control sobre el espacio libre en el disco.

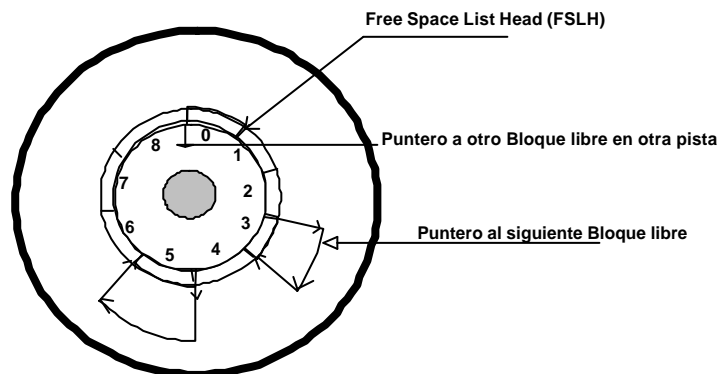


Figura 7.07 - Lista enlazada de espacio libre en disco

Este esquema no es muy eficiente ya que para recorrer la lista se debe leer cada bloque, lo que requiere un gran tiempo de E/S.

### c) Bloques de Direcciones Libres (Free Block List):

En este método a cada bloque se le asigna un número secuencial. En una **porción reservada del disco** se mantiene una lista de todos los bloques libres. Dependiendo de la capacidad del disco, con 24 o 32 bits se pueden numerar todos los sectores del disco. De esta forma, la longitud de la lista de bloques libres será de 24 o 32 veces mayor que la longitud de la tabla de bits. Por lo tanto no se puede mantener en memoria y debe ser almacenada en disco. A pesar de esto, es un método satisfactorio.

Una modificación de ésta técnica sería almacenar las direcciones de  $n$  bloques libres en el primer bloque libre disponible en el soporte. Se apuntan con un puntero a los primeros  $(n-1)$  bloques que están realmente libres. Luego en el último bloque se coloca la dirección del próximo bloque que contiene las direcciones de otros  $n$  bloques libres.

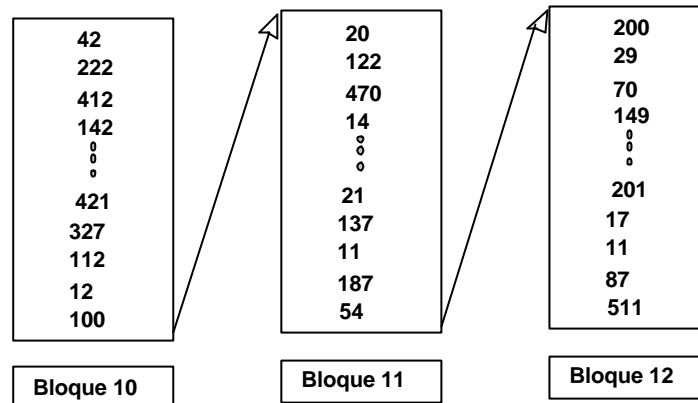


Figura 7.08 - Lista bloques libres en disco

La importancia de esta técnica es que las direcciones de un gran número de bloques libres pueden ser encontrados rápidamente.

Esta técnica toma ventajas del hecho que, generalmente, varios bloques contiguos pueden estar ocupados o libres, particularmente cuando se usa asignación dinámica en donde no se guarda la lista de  $n$  bloques libres contiguos, sino que se guarda la dirección del primer bloque libre y el número de bloques contiguos libres que le siguen. Cada entrada en la lista de espacios libres consiste entonces en una dirección en disco y un número. Cada entrada requiere más espacio que una simple dirección de disco, la lista en general será más corta, en tanto los bloques contiguos sean más de uno.

Con éste método que es una modificación del anterior, entonces se guarda en un bloque del disco tantas direcciones de bloques libres como le quepan. Así por ejemplo, con un bloque de 1024 bytes y direcciones de bloque de 2 bytes, cada bloque podría tener 512 direcciones de bloques libres.

Si se necesitaran más bloques para almacenar direcciones, la última dirección del bloque actual apunta al siguiente bloque que tiene más direcciones libres.

#### Se debe considerar los siguientes aspectos:

- El espacio en disco dedicado a la Free Block List es menor del 1% del espacio total en disco. Si el número de bloque se representara con 32 bits, la penalidad (ocupación extra) en espacio será 4 bytes por cada bloque de 512.
- A pesar que la Free Block List es muy extensa para almacenar en memoria central, existen 2 técnicas efectivas para tener en memoria central una pequeña parte de la lista:
  - La lista se puede tratar como una pila dinámica (Push Down Stack) teniendo en memoria central los primeros cientos de elementos del stack. Cuando se deba asignar un bloque a un archivo, este bloque se extrae (Pop From) de la cabeza del stack (que esta en memoria central). Análogamente cuando un archivo libera un bloque este bloque es asignado, se coloca en el tope (Put On) del stack que esta en memoria central. Las asignaciones y liberaciones no ocupan E/S (se realizan sobre el stack en memoria central) por lo tanto producen muy poco overhead. Solo hay transferencia de memoria central a disco (o viceversa) cuando la porción del stack que permanece en memoria se llena o se vacía. Esta técnica tiene casi siempre como resultado cero tiempo de acceso.
  - La lista se puede tratar como una cola (FIFO) que mantiene en memoria central solo unos cientos de elementos pertenecientes tanto a la cabeza como a la cola (tail) de la lista FIFO. Un bloque se asigna tomando el primer elemento de la cabeza de la lista y se libera agregándolo como último elemento de la porción "solo de la lista". Solo habrá una operación de transferencia (de memoria central a disco) cuando la cabeza se vacíe o la cola de libere.

Con ambas estrategias (pila o cola) un hilo en background puede ir clasificando las listas que están en memoria central para facilitar la contigüidad de las asignaciones.

**Ventajas:**

- Es más eficiente en cuanto a velocidad ya que con leer un solo bloque se conoce la dirección de una gran cantidad de bloques libres, y de esa forma no se necesitan tantas lecturas.
- Mientras más espacio haya ocupado en disco, menos espacio requiere la lista de bloques libres.

**Desventajas:**

- Cuando el disco está poco ocupado, la lista es muy grande y lleva tiempo recorrerla.

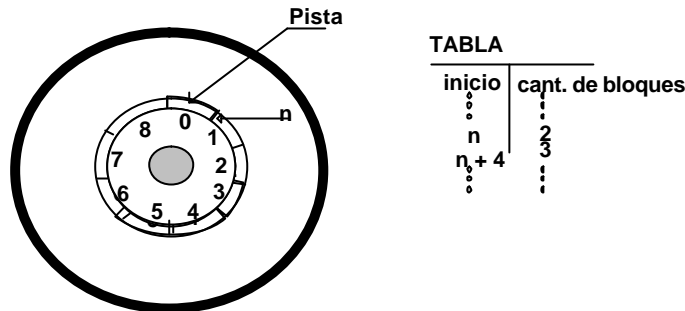
**d) Bloques de direcciones libres contiguas:**

Figura 7.09 - Lista de bloques libres

Este método es particularmente útil cuando se utiliza almacenamiento continuo o secuencial.

En vez de mantener una lista de  $n$  direcciones libres, se utiliza una lista con la dirección del primer bloque libre y el número de  $n$  bloques libres contiguos que están a continuación. De esta forma cada entrada en la lista de espacio libre consiste de una dirección y una cantidad.

**Ventajas:**

- Utiliza menos espacio de almacenamiento que el método anterior, aunque cada entrada requiere más espacio, la lista será más corta ya que generalmente habrá menos entradas.

**Desventajas:**

- Sólo es aplicable para almacenamiento continuo.

**7.3.2. Métodos de Asignación (Alocation) de espacio para los Archivos**

La implementación de un archivo es un problema del S.O.

**Asignación de Archivos**

Surgen varias cuestiones:

1. Cuando se crea un archivo nuevo. ¿Se asigna de una sola vez el máximo espacio que necesite?
2. El espacio se asigna a un archivo en forma de una o más unidades contiguas que se llaman secciones. Un tamaño de una sección puede variar desde un único bloque a un archivo entera. Que tamaño de sección debería usarse para asignar archivos?
3. ¿Qué tipos de estructura de datos o tabla se usaran para guardar constancia de las secciones asignadas a un archivo. Dicha tabla se conoce normalmente como **tabla de asignación de archivos (FAT)**.

**Asignación previa frente a Asignación dinámica:**

Una política de asignación requiere que el tamaño máximo de un archivo sea declarado un el momento de crearlo. En un número de casos, como el compilar los programas, la producción del resumen de los datos del archivo, o la transferencia de un archivo desde otro sistema por una red de comunicaciones, este valor puede estimarse. Pero en muchas aplicaciones es difícil estimar el tamaño máximo del archivo.

**Tamaño de Sección:**

La segunda cuestión es la del tamaño de sección asignada a los archivos. En un extremo, se puede asignar una sección suficientemente grande para guardar el archivo entero. En el otro extremo, se asigna el espacio en disco de bloque en bloque. Al elegir el tamaño de sección, debe haber un compromiso relativo a la eficiencia desde el punto de vista de un solo archivo frente al del sistema global.

- 1- La contigüidad del espacio aumenta el rendimiento.
- 2- Disponer de un gran número de secciones pequeñas aumenta el tamaño de las tablas necesarias para gestionar la asignación de información.
- 3- Disponer de secciones de tamaño fijo simplifica la resignación del espacio.
- 4- Disponer de secciones de tamaño variable o secciones pequeñas de tamaño fijo minimiza

la pérdida de espacio no usado provocado por la sobre asignación.

- **Secciones contiguas variables y grandes:** Esta opción ofrecerá un rendimiento mejor. El tamaño variable evitara la pérdida y las tablas de asignación de archivos serán pequeñas. El espacio es difícil de reutilizar.
- **Bloques:** Las secciones fijas y pequeñas ofrecen una flexibilidad mayor. La contigüidad se abandona y los bloques se asignan a medida que se necesitan.

Cualquier opción es compatible con la asignación previa o con la asignación dinámica. En el primer caso se asigna previamente a los archivos un grupo contiguo de bloques. En el segundo caso, todas las secciones necesarias son asignadas de una vez, Entonces la tabla de asignación de archivos permanecerá con tamaño fijo.

No esta claro que estrategia es la mejor. La dificultad de moldear estrategias alternativas esta en que intervienen muchos factores incluyendo los tipos de archivo, la pauta a los accesos a archivo, el grado de multiprogramación, etc.

Se usan tres métodos de asignación de espacio en disco: contiguo, vinculado -link- e indexado. Cada método tiene ventajas y desventajas.

### a) Asignación Contigua (Contiguous Allocation):

Este método, también llamado política de preasignación (preallocation policy), requiere que cada archivo ocupe un conjunto de direcciones contiguas o consecutivas en disco cuyo espacio debe ser declarado en la creación del archivo. Las direcciones en disco definen un ordenamiento lineal en el mismo.

La posición del archivo en el disco queda definida por la dirección del primer bloque y su longitud. Si el archivo tiene  $n$  bloques de longitud y comienza en el bloque  $b$ , entonces ocupa los bloques  $b, b+1, b+2, \dots, b+n-1$ . La entrada en el directorio para cada archivo indica la dirección de comienzo del primer bloque y la cantidad de bloques ocupados (la longitud del área asignada al archivo).

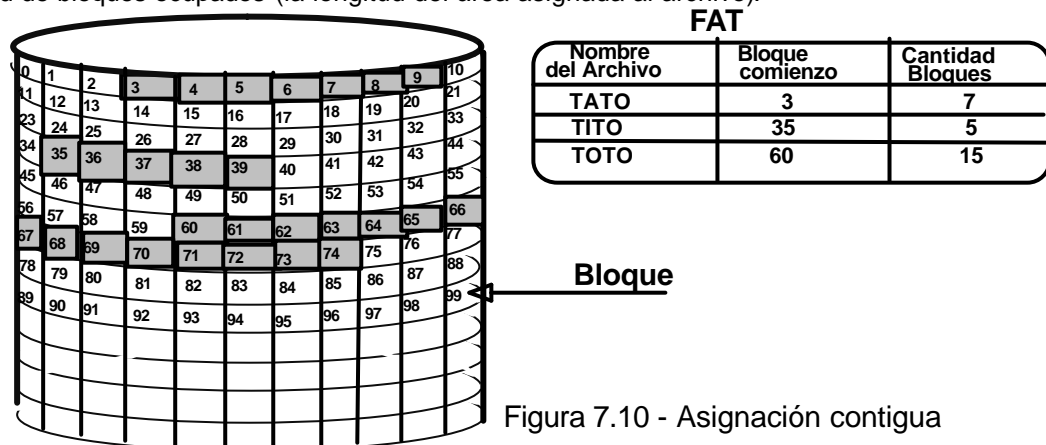


Figura 7.10 - Asignación contigua

Como hemos indicado anteriormente, con este ordenamiento, el acceso al bloque  $b+1$  luego del  $b$  normalmente no requiere movimiento de cabeza y a lo sumo solo una pista lo que incrementa la performance en las operaciones sobre el archivo orientadas a las transacciones.

El acceso a un archivo organizado con este método es muy fácil. Para un acceso secuencial, el sistema de archivo recuerda la dirección en disco del primer bloque y cuando es necesario, lee la del próximo bloque. Para un acceso directo al bloque  $i$  de un archivo que comienza en el bloque  $b$ , se accede inmediatamente al bloque  $b+i$ . Así vemos que para asignación contigua es posible tanto el acceso secuencial como el directo.

La dificultad se presenta para encontrar espacio para un nuevo archivo. Una vez implementada la tabla de bloques libres, se puede decidir como encontrar espacio para un archivo contiguo. Si el archivo a crear tiene  $n$  bloques, debemos buscar en la tabla por un espacio libre de  $n$  bloques contiguos. En el caso del mapa de bits, debemos encontrar  $n$  bits en 0 alineados; para el caso de una lista de direcciones y longitud, una longitud al menos igual a  $n$ . Este problema puede considerarse como un caso particular del problema general de la asignación dinámica del almacenamiento.

#### Ventajas:

- Es muy fácil acceder a los archivos, sólo hay que leer la zona de catálogo una vez para saber donde comienza y donde termina.
- Es ideal para accesos secuenciales ya que la cabeza no tiene que viajar mucho.

#### Desventajas:

- A medida que el disco se llena, se torna difícil encontrar espacio para un nuevo archivo. Si el

archivo a ser creado tiene  $n$  bloques de longitud debemos buscar en la free space list de la FAT los  $n$  bloques consecutivos.

- Cuando un archivo se crea, generalmente no se sabe cuantos bloques va a ocupar.
- Si un archivo tiene que crecer y no tiene más espacio consecutivo al final, hay que moverlo a una nueva localización.

Hay fragmentación externa. Es decir, puede suceder que aún habiendo espacio suficiente para un archivo, este espacio no sea contiguo, por lo tanto habrá que recurrir a un algoritmo de compactación que involucra un overhead bastante apreciable de tiempo (Considere el costo de este procesamiento).

Para aliviar un poco los problemas que crea este tipo de asignación de espacio se utilizan los mismos algoritmos de asignación que ya se vieron en el módulo de Administración de Memoria Central (First fit, Best fit, Worst fit) y que volveremos a encarar en el próximo punto.

### b) Asignación Dinámica de Almacenamiento:

El espacio de almacenamiento en disco puede verse como un gran arreglo de bloques de disco. En algún momento estos bloques son asignados a archivos y en otros están libres.

Por ende, el espacio en disco puede considerarse como una colección de segmentos libres y usados, siendo cada segmento un conjunto contiguo de bloques.

Un segmento libre se lo conoce como un *hueco*. La asignación dinámica de almacenamiento es solucionar el problema de como satisfacer un pedido de tamaño  $n$  de una lista de huecos libres. Existen muchas soluciones. El conjunto de huecos es examinado para determinar cual es el mejor a asignar. Las estrategias más comunes son las de First-fit, best-fit y worst-fit.

- **First-fit** Asignar el primer hueco lo suficientemente grande para satisfacer el pedido. La búsqueda puede iniciarse bien al comienzo del conjunto de huecos o desde el lugar donde finalizó la última búsqueda. Se detiene la búsqueda tan pronto cuando se encuentra un hueco lo suficientemente grande en la tabla para alojar el pedido.
- **Best-fit** Asignar el menor hueco que satisfaga el pedido. Para esto se debe examinar toda la tabla de huecos libres, a menos que ya este previamente ordenada por tamaño. Esta estrategia resulta en que los huecos remanentes serán los más pequeños.
- **Worst-fit** Asignar el mayor hueco disponible. También se debe examinar toda la tabla, a menos que la tabla esté ordenada por tamaño. Esta estrategia resulta en que los huecos remanentes serán los más grandes.

Las simulaciones han demostrado que tanto el First-fit como el Best-fit son mejores que el Worst-fit tanto en tiempo como en utilización de espacio. En términos de mejor utilización de almacenamiento no existe una clara superioridad entre las dos mejores estrategias, aunque el First-fit es más veloz.

Estos algoritmos sufren de fragmentación externa, es decir cuando la suma de todos los huecos es suficiente en tamaño para guardar un archivo, pero ese espacio no es contiguo.

### Compactación o Defragmentación.

Para evitar la fragmentación externa en computadores personales con disquetes o disco duro, se debe correr algún utilitario de re-empaquetado, el que básicamente copia archivos a otro medio. El soporte original es completamente liberado y entonces luego se vuelve a bajar la información asignada ahora en forma contigua. Este esquema compacta todos los huecos en uno solo. El costo de este proceso es el tiempo necesario para efectuarlo.

### Problemas de la Asignación Contigua

Otro problema con este tipo de organización es la determinación de cuanto espacio se necesita para un archivo. Al crearse el mismo, debe contarse con la totalidad del espacio libre necesario y luego asignarlo. A veces el usuario conoce la longitud, en otros casos es difícil determinar dicha longitud, entonces no siempre resultar fácil estimar el tamaño de un archivo, por ejemplo cuando se necesita copiar un archivo de salida, en general resulta dificultoso determinar la longitud.

Si se define poco espacio para un archivo, puede suceder que en determinado momento no se pueda incrementarlo. Especialmente si se usa el algoritmo Best-fit, puede ocurrir que el espacio que necesita el archivo en ambos extremos esté ocupado por otros archivos vecinos. Para esta situación existen dos posibilidades:

Primero, el programa puede terminar con un mensaje de error apropiado. El usuario debe asignar más espacio y correr nuevamente el programa, siendo estas repeticiones bastante costosas. Para prevenir esto, normalmente el usuario sobrestima el espacio necesario, resultando en considerables pérdidas de espacio en disco por la sobre-estimación.

La otra posibilidad es encontrar un hueco mayor, copiar el contenido al mismo y liberar el espacio anterior. Esta acción se repetirá mientras tanto exista espacio disponible, pero el usuario nunca se entera de lo que sucede, el programa continua aunque cada vez más lentamente.

Aun si se conoce el tamaño necesario de antemano, puede resultar ineficiente asignarlo previamente, ya que en caso de archivos que crecen muy lentamente (meses o años), tendrán su espacio reservado -sin usarlo- por largos periodos de tiempo.

### c) Asignación Enlazada o Vincular (Linked Allocation):

Cada archivo es una lista enlazada de bloques de disco. Los bloques pueden estar distribuidos en cualquier parte del soporte. El directorio contiene un puntero al primero y al último bloque del archivo entre otros punteros.

Por ejemplo, un archivo de 8 bloques que comienza en el 3, podría seguir en el bloque 6, luego el 9, el 24, 28, 36, 38 y finalmente el 42. Cada bloque contiene un puntero al siguiente bloque. Estos punteros no están disponibles para el usuario. Así si cada sector es de 512 bytes, y una dirección requiere 2 bytes, entonces el usuario ve bloques de 510 bytes.

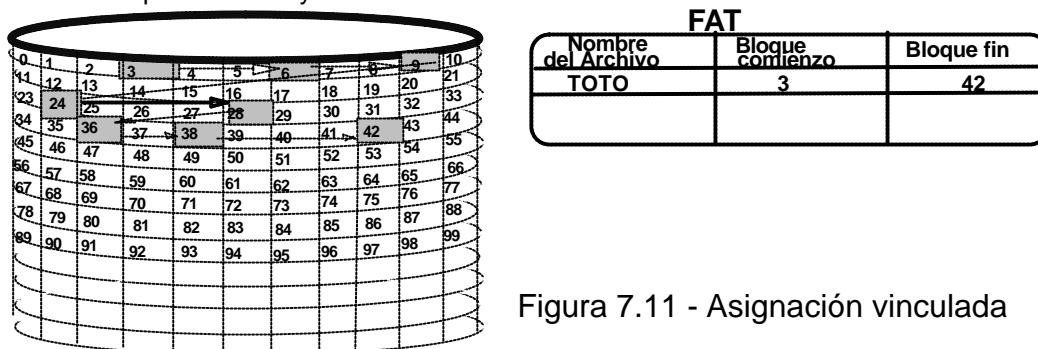


Figura 7.11 - Asignación vinculada

#### Ventajas:

- No hay fragmentación externa. Cualquier bloque libre en la free space list puede ser usado para satisfacer un requerimiento o un pedido.
- La creación de un archivo bajo éste esquema es relativamente fácil ya que solo se crea una nueva entrada al directorio. Tampoco es necesario declarar la longitud al crearlo, ya que puede crecer en tanto existan bloques libres. Por lo tanto no es necesaria la compactación.
- Un archivo puede continuar creciendo tanto como bloques libres haya en el disco.

Consecuentemente, nunca es necesario compactar el espacio en el disco (Nótese que "necesario" no significa conveniente, ya que puede ser conveniente en alguna ocasión tener todos los bloques consecutivos).

Sólo es efectivo para accesos secuenciales. Para encontrar el bloque  $i$ -ésimo de un archivo, se debe recorrer los  $i-1$  bloques anteriores (se debe ir al principio del archivo y seguir los punteros hasta el llegar al bloque  $i$ ). Cada acceso a un puntero requiere de una lectura. Por lo tanto es contraproducente implementar acceso directo con este método.

#### Desventajas:

- Los punteros ocupan espacio extra, por lo que los archivos requerirán más espacio para ser almacenados.
- Por ejemplo el espacio requerido por los punteros: si se usan 4 bytes por puntero en un bloque de 512 bytes, entonces el 0,78% del disco se usa para este propósito y no con información. Por ende cada archivo requiere de un espacio un poco mayor que el real.
- Altamente vulnerable. Como todos los bloques del archivo están vinculados mediante punteros, si se pierde un puntero, es evidente que se produce la pérdida de la información almacenada. Esto es un problema de confiabilidad.

Consideremos que pasaría si un puntero intermedio se dañara o se pierde por un algún problema con el S.O. o una falla de la unidad de discos. Esto puede generar la elección de un puntero erróneo, ya sea de otro archivo o de espacio libre. Una solución parcial es usar una lista doblemente encadenada o enlazada (doble linkeada) o almacenar en cada bloque el nombre del archivo al que pertenece y su número relativo. Por supuesto esto requiere aún más desperdicio de espacio en disco.

La asignación vincular o encadenada resuelve los problemas de fragmentación externa y de declaración de tamaño de archivos que existían en la asignación contigua. Sin embargo no soporta acceso directo, por tener punteros en bloques diseminados en todo el disco. La asignación indexada resuelve este problema trayendo todos los punteros a una sola dirección: el bloque de índices.

### d) Asignación Indexada (Indexed Allocation)

Cada archivo tiene su propio *bloque de índice (index block)*, que es un vector de direcciones de bloques en el disco. La  $i$ -ésima entrada en el bloque de índices apunta al  $i$ -ésimo bloque del archivo. El directorio contiene la dirección del index block.



Para leer el  $i$ -ésimo bloque, usamos el puntero en la  $i$ -ésima entrada del bloque de índices y así encontramos el bloque deseado. Cuando se crea un archivo, todos los punteros en el index block se inicializan en NULL. Cuando se necesita un nuevo bloque, se remueve un bloque de la free space list y su dirección se guarda en la siguiente entrada del index block.

Para grandes archivos, se puede vincular varios bloques índices. Por ejemplo, para un archivo pequeño, un index block podría tener un encabezamiento (header) donde esté el nombre del archivo y 100 direcciones de bloques. La última dirección del bloque sería NULL. Para un archivo más grande se utilizaría un puntero a otro index block, y para archivos aún más grandes se podría tener un tercer index block y así sucesivamente.

Notemos que este esquema también usa organización vinculada (linkeada) para los bloques de índices. Cada archivo debe tener, al menos uno. Entonces parecería ser que debe ser pequeño, pero si es muy pequeño, no podrá contener todas las entradas en un archivo muy grande. El bloque de índices normalmente es un bloque del disco. Por lo que para archivos grandes se usan varios bloques de índices que deben enlazarse (linkearse).

Una variante es usar un bloque de índices separado que apuntan a los distintos bloques índices que a su vez apuntan a los bloques del archivo. Para acceder a un bloque, el Sistema Operativo usa el índice de primer nivel para encontrar el índice de segundo nivel, que a su vez apunta al bloque de datos deseado del archivo.

Si caben 256 punteros en un index block, dos niveles de índices permitirán 65.536 bloques de datos, los cuales (con 1.024 bytes por cada bloque) permiten archivos de hasta 67.108.864 bytes.

Otra alternativa es guardar en el directorio por ejemplo los primeros 15 punteros del bloque de índices, lo que sirve para pequeños archivos. El UNIX usa un esquema semejante a este.

La organización indexada soporta acceso directo, sin sufrir de fragmentación. Cualquier bloque libre del disco puede satisfacer un pedido.

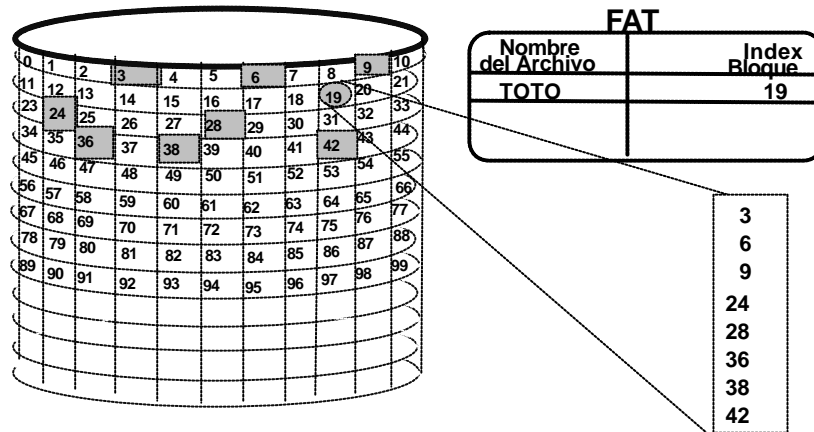


Figura 7.12 - Asignación Indexada

#### Ventajas:

- Soporta acceso directo y secuencial.
- No tiene fragmentación externa. Cualquier bloque libre del disco puede ser utilizado para almacenamiento o para índice.
- No es necesario declarar el tamaño del archivo cuando se lo crea.

#### Desventajas:

- El espacio ocupado por punteros del index block es generalmente mayor que el ocupado por la asignación enlazada. Un archivo que ocupa dos bloques desperdicia un bloque completo para punteros, con asignación enlazada el espacio perdido sería de sólo 4 bytes.

La mayoría de los archivos que normalmente se usan son relativamente pequeños. Supongamos que tenemos un archivo con solo 1 o 2 bloques. Con asignación linkeada, solo se pierde el espacio de un puntero por bloque (total de 1 o 2 punteros). En indexado, se pierde un bloque completo, aún con el archivo vacío, ya que al crear el mismo se reserva el bloque de índices.

- Para archivos muy grandes, acceder a un bloque puede involucrar muchas indirecciones con la consiguiente pérdida de tiempo. Por lo tanto este esquema es ineficiente para bases de datos grandes.

Esta forma de organización sufre de desperdicio de espacio en disco, debido al bloque de índices, el que no es necesario en asignación linkeada.

## Desempeño

La eficiencia en el almacenamiento es una característica muy importante que diferencian los distintos métodos de asignación. También difieren en el tiempo de acceso para acceder a un bloque en

disco. Este factor es relevante especialmente para sistemas chicos con discos lentos.

Una dificultad para comparar la performance de los distintos sistemas es la determinación de como serán usados. Para cualquier tipo de acceso, la asignación contigua solo requiere un acceso para conseguir un bloque. Ya que fácilmente se puede guardar la dirección del primer bloque de disco en memoria, inmediatamente se calcula la dirección en disco del  $n$ -ésimo bloque y se lo lee directamente.

Para asignación linkeada, también se puede guardar la dirección del próximo bloque en memoria y leerlo directamente. Este método es bueno para acceso secuencial, pero no para acceso directo donde llegar al  $n$ -ésimo bloque puede requerir  $n$  lecturas. Este problema indica el por que la asignación linkeada no debe usarse para aplicaciones que requieran acceso directo.

Como conclusión, algunos sistemas soportan archivos de acceso directo usando asignación secuencial y acceso secuencial usando asignación linkeada. Para estos sistemas, el tipo de acceso a utilizarse debe declararse cuando se crea el archivo.

Un archivo creado para acceso secuencial será linkeado y no podrá usarse para acceso directo. Un archivo creado para acceso directo deberá ser contiguo y podrá soportar tanto acceso directo como secuencial, pero su tamaño máximo debe declararse cuando es creado. Observemos que en este caso el S.O. debe tener algoritmos y adecuada estructura de datos para soportar ambos métodos de asignación. Los archivos pueden convertirse de un tipo a otro creando un nuevo archivo del tipo deseado y haciendo una copia. El archivo viejo se borra y el nuevo se renombra.

La asignación indexada es más compleja. Si el índice de bloques ya está residente en memoria, entonces el acceso es inmediato. Sin embargo, el almacenamiento del índice en memoria puede requerir considerable espacio. Si no es suficiente, primero debemos leer el índice y luego el bloque del archivo de datos. Para un índice de segundo grado, deben hacerse dos lecturas a índices de datos. Para un archivo muy grande, el acceso a un bloque cercano al final del archivo, requerirá leer toda la cadena de bloques de índices antes de poder acceder al bloque de datos. Así, la performance de la asignación indexada depende de la estructura del índice, el tamaño del archivo y la posición del bloque deseado.

Algunos sistemas combinan asignación contigua con indexada usando contigua para pequeños archivos (hasta 3 a 10 bloques) y automáticamente cambian a indexada y si el archivo sigue creciendo.

Para saber cuanto espacio libre hay disponible en el dispositivo, el Sistema Operativo mantiene una lista del espacio libre disponible (free space list). En esta lista se registran todos los bloques del disco que están disponibles.

Para crear un archivo buscamos en esta lista la cantidad de bloques necesarios y se los asignamos al nuevo archivo. Este espacio se recupera cuando el archivo es borrado.

Habíamos visto que el Sistema Operativo divide a los archivos en bloques de un tamaño determinado. Ahora bien, para saber en que lugar del disco se encuentra un determinado archivo, debemos conocer las direcciones de todos los bloques que lo componen. Es decir, si un archivo tiene 8 bloques de longitud, en algún lugar debe estar registrada la dirección de cada uno de los 8 bloques del mismo.

## 7.4. Concepto de Archivo

Definimos **archivo** como una colección de información relacionada entre sí, identificado por un nombre, definida por su creador que lo ve como una entidad y, generalmente, está grabada sobre algún tipo de almacenamiento auxiliar; pero desde la perspectiva del usuario un archivo es la más pequeña porción de almacenamiento secundario. Normalmente los archivos contienen información que representan programas (fuentes, objetos o ejecutables) y datos.

Conceptualmente es un conjunto de registros similares. Entidad única referenciada por su nombre. Los archivos tienen un nombre único en el sistema (en el directorio). Un archivo puede ser Creado o Borrado como una unidad (con solo mencionar su nombre) a través del file system.

Un archivo de datos puede ser numérico, alfabético, alfanumérico o binario; su forma puede ser libre (archivos de texto, imágenes, sonido, etc.) o con formato rígido en forma estricta. También puede ser: Una secuencia de bits, Bytes, Líneas, o Registros estructurados en una entidad que se reconoce mediante un nombre. Un archivo es una estructura abstracta de datos que el Sistema Operativo vincula en el aspecto lógico y en el físico como se observa en la Fig. 7.13.

El control de accesos (restricciones de uso) en general se aplica a nivel archivo. En sistemas de Time Sharing los usuarios y los programas tienen en general el acceso restringido a nivel archivo (o mas sofisticadamente a nivel bloque (conjunto de registros grabados en un mismo sector de disco), a nivel registro e incluso a nivel campo (ej: base de datos).

### 7.4.1. Tipos de Archivos

Decíamos que la información de un archivo es definida por su creador. Muchos tipos diferentes de información pueden ser almacenados en un archivo: dijimos programas fuente, objeto, o ejecutables, datos numéricos, texto, sonidos, imágenes, registro de empleados, etc.

Un archivo tiene una estructura definida por su uso y está relacionada con su tipo (ejemplo Base de Datos de Alumnos). Base de datos es un conjunto de datos relacionados cuyo aspecto esencial radica en que las relaciones entre distintos elementos de la base son explícita. La Base de Datos se diseña para ser usada por un conjunto importante de Aplicaciones. Contiene toda la información relacionada con una organización ó un proyecto. La Base de Datos consiste en varios tipos de archivos. El sistema de administración de la Base de Datos en general esta separado del Sistema operativo (es independiente del Sistema operativo, aunque hace uso del tema de archivos del Sistema operativo).

En el caso de un archivo de texto es una secuencia de caracteres organizados en líneas y páginas y un archivo fuente es una secuencia de sub-rutinas y funciones, las que a su vez están formadas por declaraciones y sentencias ejecutables. Un archivo *objeto* es una secuencia de bytes organizada en bloques comprensibles para el Editor de Enlaces del sistema; un archivo *ejecutable* es una serie de secciones de código preparadas para que luego el cargador las cargue en memoria central.

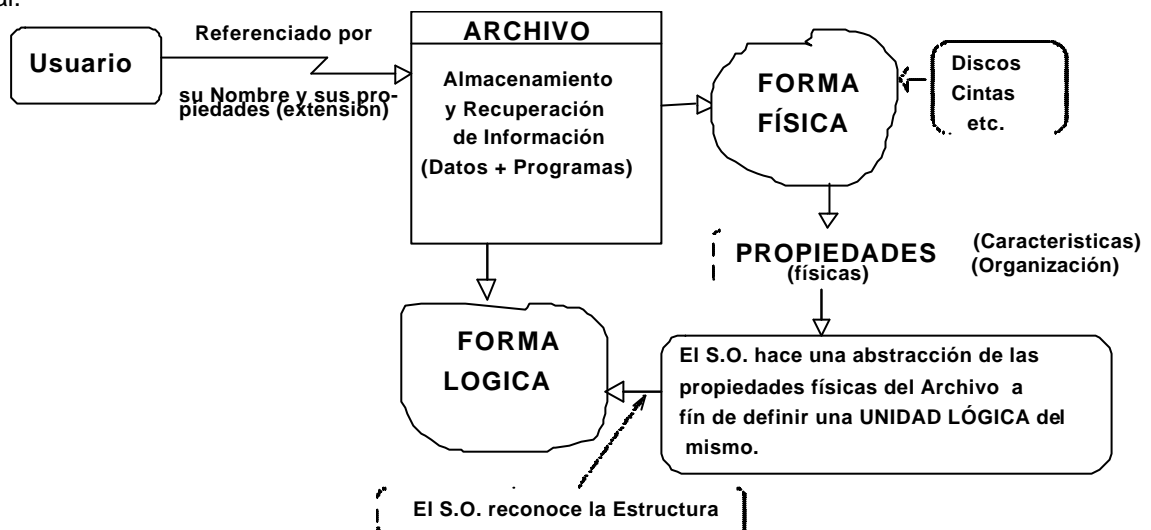


Fig. 7.13 Visiones de un archivo

Una consideración importante es que esta estructura debe ser conocida y soportada por el Sistema Operativo. Si el S.O. conoce esa estructura, puede operar con la misma de una forma adecuada. Por ejemplo, un error común es pretender imprimir un archivo objeto en binario. Este intento normalmente produce un listado con información "basura". Esto es posible evitar y no realizar la operación si el Sistema Operativo supiese de qué tipo de archivo se trata y no permitir la impresión en este caso.

La administración de archivos es uno de los servicios más visibles de un S.O.. Las computadoras pueden almacenar información en diferentes soportes físicos: disco, cintas, disquetes, etc. Cada uno de estos dispositivos tiene su propia característica y organización física.

Para el uso conveniente del sistema computacional, el Sistema Operativo provee una visión lógica uniforme de la información almacenada, ocultando las propiedades físicas y la complejidad de los dispositivos de almacenamiento y define una unidad lógica de almacenamiento: el archivo. Estos son mapeados (vinculados) por el S.O. a los dispositivos.

Una consideración importante, en el diseño de un sistema de archivos, es saber si el S.O. va a reconocer y tolerar distintos tipos de archivos. Al reconocer su tipo, este puede decidir si una operación es correctamente aplicable o no debido a que el Sistema de Archivo es "Propietario" de ese S.O..

TIPO DE ARCHIVO	EXTENSIÓN USUAL	FUNCIÓN
ejecutable	exe, com, bin, o ninguna	programa en lenguaje de máquina, listo para ejecutar
objeto	obj, o	lenguaje de máquina compilado, sin "linker"
código fuente	c, p, pas, f77, asm, a	código fuente en varios lenguajes. Ejemplo c ⇒ lenguaje C
batch	bat, sh	comandos del interprete de comandos
texto	txt, doc	datos textuales, documentos

procesador de texto	wp, tex, rrf, doc etc.	formatos varios de procesador de texto
biblioteca	lib, a	bibliotecas de rutinas para programadores
impresión o imagen	ps, dvi, gif	archivo ASCII o binario en formato para impresión o para ver por pantalla
archivo	arc, zip, tar	archivos relacionados agrupados en uno solo, a veces comprimidos para su almacenamiento

**Tabla 7.1** Los archivos y sus extensiones.

Una técnica común es incluir el tipo como *extensión* del nombre separándolo con un punto, a partir de esto, el sistema indica en la extensión el tipo y las operaciones posibles de realizar. En la **Tabla 7.1** se presentan algunas extensiones utilizadas en MS-DOS; pero los programas de aplicaciones también utilizan las extensiones para indicar sus archivos de trabajo; por ej. el procesador de texto Word Perfect utiliza archivos con extensión “.wp”.

Podemos ver otro ejemplo de la utilización de tipos de archivos en algunos Sistemas Operativos, si un usuario trata de ejecutar un programa objeto que fue editado luego de su producción, el programa fuente es recompilado automáticamente asegurando la ejecución del programa actualizado. Para poder implementar este mecanismo, el S.O. debe poder discriminar entre el objeto y el fuente, chequear el momento de su última modificación y el lenguaje utilizado (para usar el compilador adecuado).

En el sistema operativo APPLE MACINTOSH cada archivo tiene un tipo como ser “texto” o “imagen”; además cada uno cuenta con un atributo de creación conteniendo el nombre del programa que lo creó. Cuando un usuario abre el archivo (con un doble-click en el icono representativo) el programa creador es invocado automáticamente y se carga el archivo, listo para editarse.

El sistema operativo UNIX no está preparado para brindar tales características porque utiliza un limitado *número mágico*, que almacenado al principio de algunos archivos indica aproximadamente el tipo del archivo. Como no todos los archivos tienen número mágico no se puede brindar este servicio. En UNIX no se registra el nombre del programa creador y se permiten extensiones indicativas; pero las extensiones no dependen del S.O. sino que el significado es asignado por el usuario creador.

En general los archivos representan programas y datos. Pueden tener un formato rígido tal como los archivos de una base de datos o libre tal como los archivos de texto.

## 7.4.2. Atributos de los Archivos.

Entonces, un archivo se identifica con un *nombre y una extensión* y es referenciado por el mismo. La extensión contiene propiedades tales como tipo, hora de creación, nombre del creador, longitud, etc.

A los archivos se les asigna un nombre para facilitar su manipulación por parte del usuario, que hace referencia a él por ese nombre (normalmente una secuencia de caracteres como ejemplo “toto.c”); algunos sistemas diferencian entre letras mayúsculas y minúsculas. Otros no lo hacen. Cuando a un archivo se le asigna un nombre, este se vuelve independiente del proceso, usuario y hasta del sistema que lo creó. Pudiendo, por ejemplo, ser editado por otro usuario, grabado en un disquete y leído en otro sistema; todo esto simplemente con su nombre original.

Algunos atributos típicos de un archivo, los cuales pueden variar de un sistema a otro, son:

- **NOMBRE:** El nombre simbólico es información con sentido para el usuario.
- **TIPO:** Información necesaria para aquellos sistemas que soportan diferentes tipos de archivos.
- **UBICACIÓN:** Un puntero al dispositivo y hacia las locaciones que ocupa el archivo.
- **TAMAÑO:** El tamaño actual (en bytes, palabras o bloques) y el máximo tamaño posible.
- **PROTECCIÓN:** Información e control de acceso que determina quien puede leer, modificar y ejecutar el archivo.
- **TIEMPO, FECHA e IDENTIFICACIÓN DEL USUARIO:** Esta información se guarda en los siguientes momentos: creación, última modificación y último uso; son datos de uso para protección, seguridad y monitoreo del uso.

Todos estos datos acerca del archivo son almacenados en la estructura el directorio de archivos del soporte (almacenamiento auxiliar). Cada archivo tiene una entrada en el directorio que puede ocupar entre 16 y 1000 bytes, dependiendo del Sistema Operativo, por lo que se dice que es también “propietario” de cada uno de ellos. La palabra propietario se debe entender como particular de un S.O. y no compatible con otros.

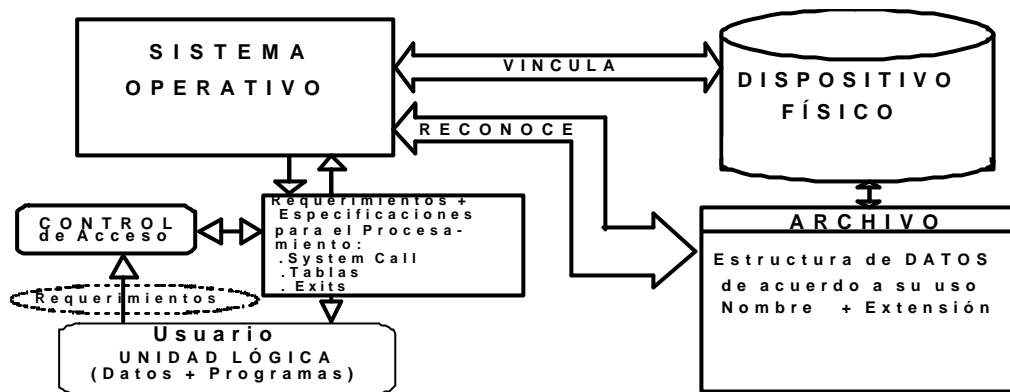


Fig. 7.14 Relación de vinculación y reconocimiento de un archivo

Para la mayoría de los usuarios, el sistema de archivos es el aspecto más visible de un S.O. Este implementa el concepto abstracto de la estructura de datos, administrando los dispositivos de almacenamiento, tales como discos y cintas.

Los archivos están organizados en directorios para hacer más fácil su uso. Estudiaremos los distintos tipos de estructura de directorios más adelante en este módulo.

Finalmente, cuando múltiples usuarios tienen acceso a archivos, es deseable un control de quienes y con que condiciones pueden accederlos. Este control es conocido como protección de archivos.

En particular, los usuarios efectúan sus requerimientos sobre la estructura abstracta de datos al S.O. quien no solo controla los aspectos de seguridad y protección en los accesos, sino reconoce como suyo al archivo y lo vincula con el dispositivo físico que contiene el soporte donde está almacenado como se observa en la Fig. 7.14.

### 7.4.3. La estructura de la Información

Las computadoras solo manejan (operan, transmiten, almacenan, etc.) los datos en forma binaria (bits). Los programas y el ser humano manejan conjuntos de símbolos o caracteres que la máquina debe armar con un conjunto de bits. Para ello utiliza un código que efectúa esa transformación en forma ordenada bajo las Reglas con que se manipula el código (Ejemplo el código ASCII - American Standard Code for Information Interchange- utiliza 7 bits). En la figura 7.15 describimos los distintos componentes de una estructura de información.

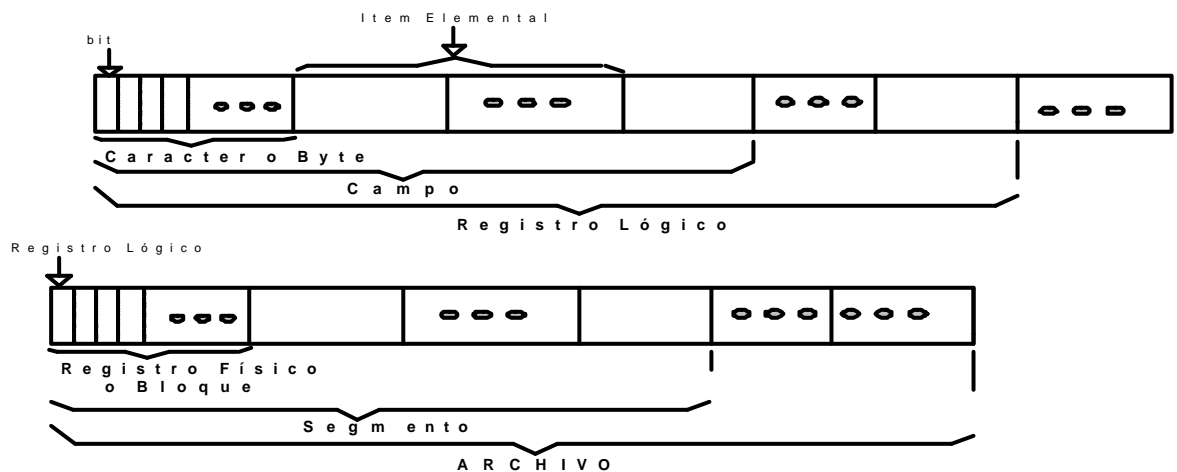


Fig. 7.15 Distintas estructuras de información

Un conjunto de caracteres relacionados entre sí y tomados como una unidad se denomina campo (data field). Un ejemplo es el campo fecha ya que es la mínima unidad de dato que se puede referenciar como fecha. En el caso de una Base de Datos es un componente designable de un segmento de base de datos.

El **Campo** se considera un elemento básico de información. Contiene un solo valor por instancia. Se caracteriza por su **Longitud y el Tipo de dato** (numérico, alfabético, etc). La longitud puede ser **Fija o Variable**. En este último caso, consiste en 2 o 3 subcampos: Valor, Nombre del campo y Longitud. La longitud del campo se indica con símbolos especiales que demarcan cada campo.

Una clave (key) es un campo de identificación único que contiene un registro lógico en un almacenamiento, que se utiliza para identificar, acceder o localizar al registro para operarlo o clasificarlo en un orden determinado.

Un campo puede estar integrado por un conjunto de ítems elementales como ser el día, el mes o el año de un campo fecha. Distintos campos relacionados y referidos a una misma entidad, concepto o sujeto se agrupan en un registro (record) que se trata como una unidad lógica (registro lógico). Varios registros lógicos se pueden agrupar en un registro físico o bloque para efectuar un procesamiento o transferencia o movimiento de datos.

**Registro** es Conjunto de campos relacionados, tratados como una unidad por los programas que hacen uso del archivo. La **longitud del registro puede ser Fija** (conjunto fijo de campos de longitud fija) o **Variable** (conjunto variable de campos de longitud fija o conjunto fijo de campos de longitud variable).

Un conjunto finito de registros homogéneos relacionados entre sí, se agrupan en una entidad lógica llamada archivo (file). Los archivos suelen agruparse en una tabla de símbolos, con identificadores y referencias correspondientes a tipos de datos, llamada directorios (directory).

Así se pueden agrupar los archivos en Bases de Datos (Data Base) o Banco de Datos (Data Bank).

#### 7.4.4. Archivos Mapeados a Memoria.

La idea es la siguiente: supongamos la existencia de dos nuevas llamadas al sistema (system calls), *map* y *unmap*. La primera utiliza el nombre del archivo y una dirección virtual y realiza una asociación del archivo con la dirección virtual en el espacio de direcciones.

Por ejemplo, supongamos que un archivo TATO de longitud 64 KBy es asociado con la dirección virtual 512 K; a partir de esto cualquier instrucción de máquina que lea el contenido del byte en 512 K obtiene el byte 0 del archivo. Análogamente, al escribir en la dirección 512 KBy +100 se modifica el byte 100 del archivo. Al finalizar el proceso el archivo modificado se encuentra en el disco tal como si hubiese sido alterado a través de una combinación de *seek* y *write* que veremos más adelante.

Por ejemplo si estamos trabajando con una administración de Memoria Paginada por Demanda ocurre que las tablas internas del sistema se modifican permitiendo que el archivo sirva como copia de seguridad de sí mismo. Una lectura en 512 KBy provoca un *page fault*, recuperándose la página 0 del archivo. Análogamente la escritura en 512 KBy + 100 provoca un *page fault*, recuperándose la página con dicha instrucción, luego se lleva a cabo la escritura en memoria. Si el algoritmo de reemplazo de páginas elimina la página, se escribe en el lugar adecuado del archivo; por lo tanto al terminar el proceso todas las páginas asociadas y modificadas se re-escriben en sus correspondientes archivos.

#### 7.4.5. Nombres de Archivos

En MS-DOS el nombre puede tener hasta ocho caracteres, un punto y una extensión de hasta tres caracteres. Por ejemplo, un usuario elige para un archivo, de ayuda documentada para el uso de cierto programa, el nombre "ayuda.doc".

En UNIX el nombre de un archivo puede tener hasta 14 caracteres en la mayoría de los system V y un máximo de 255 caracteres en todos los sistemas basados en Berkeley, donde los únicos caracteres no permitidos son: la barra (/), blanco, menos y guión.

#### 7. 4.6. La estructura de un Archivo

Los tipos de archivo también se usan para indicar su estructura interna. Por ejemplo los archivos fuente y objeto tienen cierta estructura que coincide con lo esperado por los programas que los leen, es más, hay ciertos archivos que deben cumplir con la estructura requerida por el S.O. para que éste los comprenda. Como ser el caso de los archivos ejecutables, donde de acuerdo a su estructura el S.O. determina su punto de carga en memoria y donde se encuentra la primera instrucción.

Algunos S.O. toleran una cantidad determinada de estructuras de archivos y cuentan con operaciones especiales de manejo de archivo para estas estructuras, por ejemplo el sistema VMS de DEC (Virtual Machine System de Digital Equipment Company) tiene un sistema de archivos con tres estructuras definidas.

Una de las desventajas del manejo de varias estructuras es que el tamaño del Sistema de Gestión de Archivos se torna inmanejable. Si un S.O. define cinco estructuras distintas, necesita contar con el código para soportar esas estructuras. Además todos los archivos deben ser definidos en uno de los tipos de archivo soportados por el sistema. Las nuevas aplicaciones que necesitan información estructurada de manera diferente a las soportadas por el sistema se encontraran con varios problemas.

Supongamos que un determinado S.O. soporta dos tipos de archivos: texto y binarios ejecutables. Ahora, si un usuario quisiera definir un archivo encriptado podría encontrarse con que ninguno de los tipos soportados es apropiado para su necesidad, teniendo que optar entre un uso

inapropiado del mecanismo ofrecido por el sistema o modificar su técnica de encriptación.

Algunos Sistemas Operativos imponen un número mínimo de estructuras de archivos este enfoque fue adoptado por UNIX, MS-DOS y otros. UNIX considera cada archivo como una secuencia de bytes de 8 bits y no hace ninguna interpretación de esos bits. Este esquema permite una máxima flexibilidad pero poco soporte, ya que cada aplicación debe incluir su propio código de interpretación para un archivo de entrada. Sin embargo todos los Sistemas Operativos deben soportar por lo menos una estructura, la de archivo ejecutable, así el sistema puede cargar y ejecutar los programas.

Otro ejemplo de mínima cantidad de estructuras es el S.O. MACINTOSH, el cual opera archivos con dos partes: *resource fork* y *data fork*. El resource fork contiene información de interés para el usuario. Por ejemplo, éste guarda las etiquetas de los botones mostrados por el programa. Un usuario distinto necesita reetiquetar esos botones en su propio lenguaje, y el sistema le brinda herramientas para realizar la modificación de los datos del resource fork. El data fork contiene el código del programa y los datos: los contenidos tradicionales de un archivo. Si quisiéramos cumplimentar la misma tarea en UNIX® o MS-DOS® tendríamos que cambiar y recompilar el código fuente. Lo importante de este ejemplo es rescatar que es provechoso para un Sistema Operativo soportar estructuras de uso frecuente, restándole esfuerzo al programador.

Pocas estructuras son inconvenientes para la programación, mientras que muchas incrementan el tamaño del Sistema Operativo y confunden al programador.

#### 7.4.7. Estructura Interna

Internamente, puede ser complicado para el S.O. ubicar un desplazamiento (*offset*) determinado dentro de un archivo. Como los sistemas de discos normalmente tiene el tamaño de bloque definido de acuerdo con el tamaño de un sector (o múltiplos), todas las operaciones de Entrada - Salida se realizan de a un bloque (registro físico) y todos los bloques son del mismo tamaño, es casi seguro que el tamaño del registro físico no concuerde exactamente con el del registro lógico deseado. Al variar el largo del registro lógico se puede aplicar como solución un empaquetado de registros lógicos en bloques físicos como se explicó anteriormente.

Por ejemplo en UNIX se definen a todos los archivos como un simple flujo de bytes. Cada byte es individualmente direccionable por su desplazamiento desde el comienzo (o fin) del archivo, dando como resultado un registro lógico de 1 byte. El sistema de archivos empaqueta y desempaqueta bytes en bloques del disco (512 bytes por bloque) de acuerdo a las necesidades.

El tamaño del registro lógico, del bloque físico y la técnica de empaquetado determinan cuantos registros lógicos hay en cada bloque físico. El empaquetado puede ser realizado por la aplicación, o bien, por el S.O.; en ambos casos un archivo debe ser considerado como una secuencia de bloques ya que las funciones básicas de Entrada - Salida operan en estos términos. La conversión de registros lógicos a bloques físicos es un problema de software con solución simple.

Como el espacio de disco se asigna de a bloques, generalmente una parte del último bloque es desaprovechada por **fragmentación interna**. Cuando más grande es el tamaño del bloque, mayor es la fragmentación interna.

### 7.5.- Organización lógica y acceso a los archivos (file organization and access)

#### 7.5.1.- Organización lógica

Consideramos el término **organización lógica de archivos** para referirnos a la estructura lógica de los registros determinada por la manera en que se accede a ellos. La **organización física** del archivo en el almacenamiento secundario depende de la estrategia de agrupación (ablocamiento) y de la estrategia de asignación del espacio físico para el archivo.

Para seleccionar una organización lógica de archivos hay diversos criterios que son importantes:

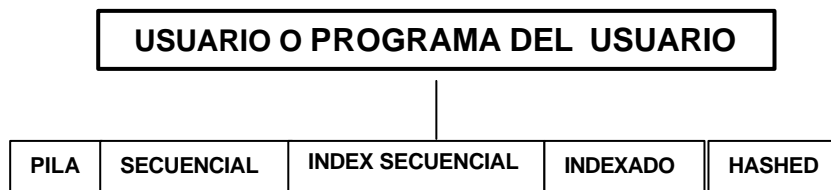


Figura 7.16 Organización lógica y accesos

1 Acceso rápido a los registros para recuperar o guardar la información

- 2 Facilidad de actualización.
- 3 Economía de almacenamiento.
- 4 Mantenimiento sencillo y simple
- 5 Confiabilidad para asegurar la confianza de los datos

La prioridad relativa de estos criterios dependerá de **LAS APLICACIONES** que usarán el archivo.

Ej: si a un archivo siempre lo voy a acceder en forma batch y en su totalidad, entonces el poseer un acceso rápido para recuperar un registro va a ser de mínima importancia.

Un archivo almacenado en un cd-rom jamás será actualizado.

El punto 2 no interesa para nada. A veces los criterios 1 y 3 son contrapuestos, ej: para 3 hay que reducir la redundancia de datos a cero, por otro lado la redundancia es el único medio para incrementar la velocidad de acceso a los datos (creando índices). Hay libros enteros dedicados a organización de archivos y métodos de acceso asociados.

La cantidad de alternativas de organizaciones lógicas de archivos que se han implementado o propuesto en distintos S.O. es variado. Cada uno propone ventajas y desventajas en sus implementaciones. La mayor parte de las estructuras empleadas en los sistemas reales se encuadran en una de estas categorías o puede implementarse como una combinación de estas. Por sencillez veremos brevemente 5 organizaciones fundamentales:

1. Pilas (The pile)
2. Archivos secuenciales (sequential file)
3. Archivos Secuenciales indexados. (indexed sequential file)
4. Archivos indexados.(indexed file)
5. Archivos directos o de dispersión (director hashed, file).

### **Pilas**

La forma menos complicada de organización de archivos puede denominarse la pila. Los datos se recolectan en el orden en que llegan. Cada registro consiste en una ráfaga de datos. El propósito de la pila es acumular la masa de datos y guardarlo.

Los registros pueden tener distintos campos o campos similares en distinto orden. Por lo tanto cada campo se debe describir a si mismo: incluyendo un nombre y un valor. La longitud de cada campo debe estar indicada implícitamente por delimitadores, explícitamente incluida como un subcampo o conocida por defecto de acuerdo al tipo de campo. Como no hay una estructura para el archivo de la pila, el acceso a registros se hace por medio de búsqueda exhaustiva (en todo el archivo). Es decir, si se quiere encontrar un registro que contiene un campo particular con un valor determinado, es necesario examinar cada registro de la pila hasta que se encuentre el registro deseado o se haya recorrido el archivo completo (la pila). Si deseamos encontrar todos los registros que contengan un campo en particular o contengan un valor particular en dicho campo, debemos buscar en todo el archivo.

Los archivos de pilas se aplican cuando los datos se recogen y almacenan antes de procesarlos o cuando no son fáciles de organizar. Este tipo de archivo usa bien el espacio cuando los datos almacenados varían en tamaño y en estructuras. Este tipo de archivos no se adapta a la mayoría de las aplicaciones. Al ser una organización adecuada para búsquedas exhaustivas, es fácil de actualizar (se graba encima). Fuera de estos usos es una organización inadecuada para la mayoría de las aplicaciones.

### **Archivos Secuenciales**

Es la organización de archivos más común. Se emplea un formato fijo para los registros. Todos los registros consisten del mismo del mismo número de campos de longitud fija, colocados en el mismo orden dentro del registro. Como se conocen la longitud y la posición de cada campo, sólo se necesita almacenar los valores de cada campo (para cada registro); el nombre del campo y la longitud de cada uno son atributos de la estructura del archivo.

Un campo particular (generalmente el primero en cada registro), es conocido como el **campo clave** que identifica unívocamente al registro; así, los valores de la clave para registros diferentes son siempre diferentes. Además, los registros se almacenan en secuencia por clave (orden alfabético o numérico). Los archivos secuenciales se utilizan normalmente en aplicaciones de proceso por lotes (batch), que involucran el tratamiento de todos los registros del archivo (ej: facturación, sueldos, etc.). La organización secuencial de archivos es la única que permite un almacenamiento sencillo tanto en cinta como en disco.

Para las aplicaciones interactivas que incluyen peticiones o actualizaciones de registros individuales, los archivos secuenciales ofrecen un rendimiento pobre ( para acceder a un registro cualquiera voy a tener que acceder en promedio a  $(N/2 - 1)$  registros previos, siendo N la cantidad total de registros contenidos en el archivo). Los accesos requieren del acceso secuencial de los registros hasta obtener un apareamiento de claves (KEY MATCH). Si fuera posible traer a memoria todo el archivo de una sola vez (casi imposible para archivos voluminosos como los archivos maestros), deberíamos efectuar N lecturas (1 por cada registro) y luego  $\lg 2N$  búsquedas (si uso búsqueda binaria). Por lo tanto acceder a un solo registro en un archivo secuencial grande va a generar demoras y procesamiento considerables.



El agregado de registros también es problemático. Generalmente un archivo secuencial se almacena sencillamente mediante el ordenamiento secuencial de los registros dentro de los bloques, por lo tanto la organización física el archivo refleja la organización lógica del archivo. Generalmente el agregado de registros se efectúa sobre un archivo pila auxiliar. Y de tanto en tanto la pila se intercala con el archivo de entrada obteniéndose un archivo de salida actualizado, con los registros de la pila colocados en el orden correcto.

Una alternativa es organizar el archivo secuencial como una lista encadenada. En cada bloque físico almacena uno (o varios) registros lógicos. Y cada bloque físico (en el disco) contiene un puntero hacia el bloque lógicamente siguiente. La inserción de nuevos registros involucrará una manipulación de punteros, sin requerir que el nuevo registro ocupe una determinada posición física del bloque que la contenga. Así agrego una nueva facilidad de uso, pero a costa de procesamiento adicional y sobrecarga (overhead).

#### **Archivos Secuenciales indexados**

Un método popular para superar las desventajas de los archivos secuenciales (sobremana para acceder a un solo registro o unos pocos a la vez) es el del archivo secuencias indexado.

Mantiene las características básicas de los archivos secuenciales. Los registros se organizan en una secuencia basada en un **campo clave**, pero se añaden dos características nuevas: un archivo de **índices** para soportar los accesos aleatorios y un **archivo de desbordamiento** (overflow). El índice proporciona una capacidad de búsqueda (lookup) para llegar rápidamente a la vecindad del registro deseado. Cada entrada en el índice posee la clave más alta del bloque lógico, seguida de la dirección física (CCHSS) del primer bloque físico que contiene a ese bloque lógico. Para encontrar un campo específico ( $\lg_2 N$  con N cantidad de entradas en el índice, ordenado en forma ascendente por "llave mas alta del bloque lógico asociado"), se recorre el índice hasta encontrar en tabla la primer clave mayor o igual al campo clave buscado. Obtenida la dirección del bloque lógico correspondiente; se lee todo ese bloque lógico (a veces varios bloques físicos). Y luego se busca secuencialmente (en Memoria Central) en el bloque lógico hasta encontrar el registro con la clave pedida (por ej. en Cobol, si no se lo encuentra, se obtiene una señal de Clave Inválida como salida de error). Ese registro lógico se le pasa al programa que lo pidió. Consideremos un archivo secuencial con 1.000.000. Para acceder un registro al azar en promedio deberá acceder a 500.000 registros. Supongamos tener ahora ese mismo archivo pero con una organización Secuencial Indexada con un índice de 1000 entradas. Encontrar la clave buscada recorriendo secuencialmente el índice llevará en promedio 500 búsquedas (si se usa búsqueda binaria  $\lg_2 1000=10$  pues  $2^{10}=1024$ , se empleará en promedio 10 búsquedas; pero requiere tener ordenada en forma ascendente el archivo de índices).

Ubicado el bloque lógico (que contendrá 1000 registros), implica que la búsqueda secuencial del registro con la clave deseada llevará 500 búsquedas. En total  $500+500=1000$  búsquedas en lugar de las 500.000 originales. Mil veces menos. Para mejorar la eficiencia se agrega un nivel de índices (otro archivo de índices). En este caso al 1er archivo de índices se llamará índice de 2do nivel y se construirá el nuevo archivo (índice de 1er. Nivel) de la siguiente manera:

**entrada(i)= entrada - 2do nivel ((i+1)\*100).**

El archivo de 1er nivel contendrá 100 entradas. Y cada entrada permitirá recorrer el tramo  $(i*100) < j \leq (i+1)*100$  de la tabla de 2do nivel.

La tabla de 2do. nivel contendrá 10.000 entradas (cada bloque lógico ahora contendrá 100 registros en lugar de 1000).

Si sobre cada tabla se busca secuencialmente ahora en promedio se tendrá  $50+50=100$  búsquedas en índices, y luego si 50 búsquedas secuenciales en el bloque lógico de registros. Por lo tanto en lugar de 1000 búsquedas ahora se emplea solo 150. Pero se ha agregado otro archivo de índices y se ha incrementado 10 veces la longitud del archivo secundario. Si se usa búsqueda binaria en los índices se tendrá en promedio  $5,6+5,6+50=61,2$  búsquedas para encontrar (o no) una clave determinada.

**Agregado de registros:** cada registro en el archivo principal tiene un campo adicional (no visible para los usuarios) que actúa como puntero al área de desbordamiento. Cuando se agrega un registro en el área de desbordamiento el puntero del registro que lo precede (en general está en el área principal) se carga con la dirección del bloque físico que contiene el registro agregado. Y de ser necesario se actualiza también las tablas de índices (si la clave del nuevo registro es la más alta del "bloque lógico" que está en el archivo principal). Si el "precedente" del registro agregado está a su vez en el área de desbordamiento, entonces se carga el puntero del precedente con la dirección física del bloque donde se almacenó el registro recién insertado. Cuando en el área de desbordamiento se tiene muchos registros, se debe "reorganizar" todo el archivo; o sea que se debe intercalar los registros del área de desbordamiento con los registros del área principal. Se hace en un proceso batch. Se guarda el index secuencial original en una cinta (ordenando el archivo resultante por la clave principal del index

secuencial). Y luego se "genera" un nuevo archivo index secuencial cargando todos los registros de la cinta en un área principal lo suficientemente grande como para tener una nueva área de desbordamiento vacía. Como se ve esta organización preserva la organización secuencial, y permite un acceso bastante rápido a registros individuales del archivo. Cuando acceso secuencialmente al archivo, los registros del área principal se acceden secuencialmente hasta llegar a un registro con puntero activo. Allí paso a operar en el área de desbordamiento hasta que deba encadenar con un puntero **NULO**. En ese momento vuelvo al área principal donde estaba y continúo operando en forma secuencial hasta encontrar otro puntero **ACTIVO**, con lo que repito el proceso mencionado anteriormente. Las bajas de registros se **MARCAN** en un campo específico no accesible por el usuario. Estos registros **NO** se graban en la cinta cuando se está reorganizando el archivo, con lo cual luego de la reorganización desaparecen del archivo.

La actualización de cualquier registro no posee inconvenientes pues los registros son de longitud fija. Los campos CLAVE PRINCIPAL no se pueden "**actualizar**". Debo dar una baja y luego un alta.

### Archivos Indexados

El index secuencial es una leve mejora del secuencial. Sólo mejora el rendimiento cuando se accede un registro al azar por la clave principal, un solo campo del archivo.

Cuando es necesario buscar un registro basándose en algún otro atributo distinto del campo clave principal, ambas formas de archivo secuencial no son adecuadas. En algunas aplicaciones se hace necesaria esta posibilidad (acceder un registro al azar a través no de un único campo sino de varios posibles). Para implementar esta flexibilidad se necesita de una estructura que emplee múltiples índices, un archivo índice por cada tipo de campo que permita efectuar una búsqueda.

En esta organización se abandonan los conceptos de secuencialidad y de claves únicas. Los registros sólo se acceden a través de sus índices. Como consecuencia ya no hay restricciones en cuanto a la colocación de los registros mientras exista al menos un puntero (en al menos uno de los índices) que referencie a cada registro. Pueden emplearse registros de longitud variable.

Existen dos tipos de índices: un **índice exhaustivo** que contiene una entrada para cada registro del archivo principal y se organiza en sí mismo como un archivo secuencial y un **índice parcial** que contendrá entradas a los registros donde esté el campo de interés (pues en registros de longitud variable el campo puede a veces no existir en algunos registros). Cuando se añade un registro al archivo principal, físicamente se coloca al final del mismo, y luego se actualizan todos los índices.

Los archivos indexados tienen un rendimiento bajo cuando se los accede exhaustivamente. Se los usa principalmente cuando las aplicaciones tienen tiempos críticos de respuesta, por ej: sistema de reserva de pasajes, de control de inventarios, etc.

**Archivos directos (o Hashed: reformulado):** explotan la capacidad de los discos para acceder directamente a cualquier bloque de dirección conocida. Al igual que las organizaciones secuencial y secuencial indexada, se requiere que cada registro posea un campo clave (key field). Aquí no hay concepto de ordenación secuencial. El archivo directo emplea una reformulación del valor de la clave. En una organización directa se emplea un archivo de desbordamiento para colocar los "sinónimos" en forma encadenada. Los archivos directos son usados a menudo donde se necesita un acceso muy rápido, donde se usan registros de longitud fija y donde siempre se accede a los registros de una vez. Ejemplo: directorios, tablas de precios, planificadores y listas de nombres.

## **7.5.2. MÉTODOS DE ACCESO.**

Para utilizar la información almacenada en los archivos, se debe leer y traerla a memoria. Existen varias formas de acceso a archivos. La elección de un método apropiado de acceso para una determinada aplicación, es uno de los principales problemas de un diseñador. Algunos S.O. permiten un único método de acceso mientras que otros, soportan diferentes tipos de acceso.

Definimos a un método de acceso como un conjunto de rutinas y tablas que posibilitan acceder a la información según un esquema lógico determinado y se ocupa de la interfase del programa del usuario con el módulo de Entrada / Salida en cuanto al buffering, el blocking y la independencia de los dispositivos. En la **Figura 7.17** presentamos un esquema de los métodos de accesos.

Los métodos de acceso solo procesan aquellos archivos que se ajustan a ciertas normas de organización y al uso de dispositivos cuyas características puede manejar. Los otros archivos deberán ser atendidos por el S.O..

Los accesos lógicos pueden ser:

- 1) Acceso secuencial.
- 2) Acceso directo
- 3 Acceso indexado

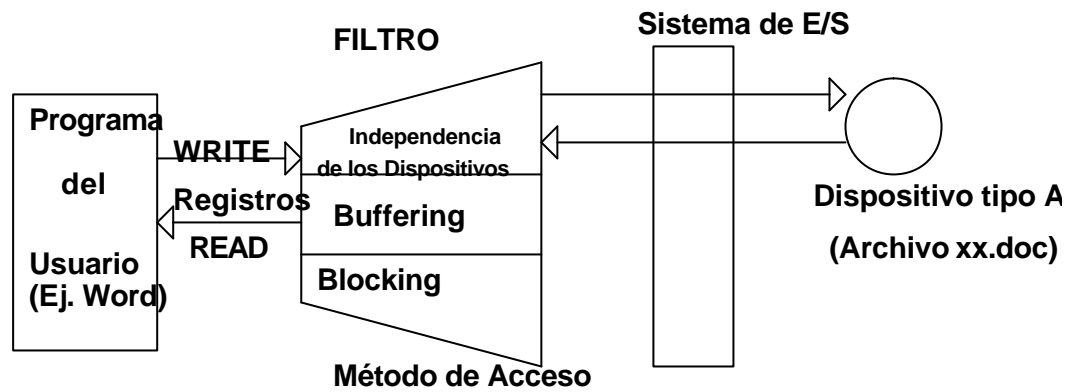


Fig. 7.17 Los métodos de acceso

## a) Acceso Secuencial

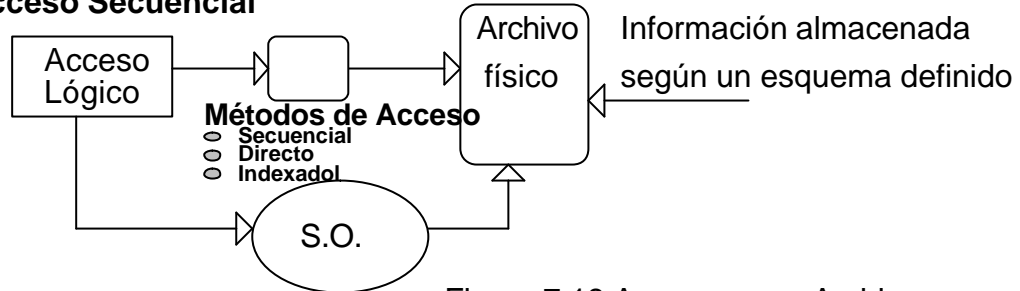


Figura 7.18 Accesos a un Archivo

Es el método más simple, la información del archivo se procesa ordenadamente y de a un registro por vez. Es el método más común, lo usan normalmente los compiladores y editores.

El grueso de las operaciones sobre el archivo son lecturas y escrituras. Un *read* lee la próxima porción del archivo y automáticamente avanza el puntero de archivo, el cual rastrea la ubicación de la próxima E/S como se indica en la figura 7.19. Similarmente, un *write* agrega datos al final del archivo y avanza al nuevo fin de archivo. Así también un archivo puede posicionarse al principio, y, en algunos sistemas, un programa puede saltar hacia adelante o hacia atrás n registros.

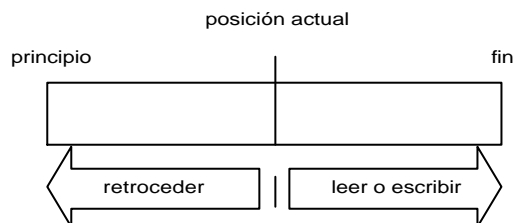


Figura 7.19 posición de los punteros

Una operación de lectura lee una porción del archivo y automáticamente avanza el puntero del archivo. Lo mismo sucede en la escritura (se avanza al nuevo fin de archivo). En algunos sistemas un programa puede ser capaz de saltar hacia adelante o hacia atrás una cantidad n de registros.

Este esquema se lo conoce como acceso secuencial y se basa en el modelo de cinta de un archivo y trabaja tanto en dispositivos de acceso secuencial como en los de acceso aleatorio.

**Resumen** de las principales características del Acceso secuencial:

- Forma más básica de organización.
- Todos los registros tienen el mismo número de campos y todos los campos tienen igual tamaño y están colocados en el mismo orden.
- El primer campo suele tener información única y ser el campo clave.
- Los registros se almacenan en orden según la clave.
- Es el único sistema que se puede almacenar tanto en disco como en cinta.
- Tiene poca eficacia para modificaciones individuales, pero puede ser muy eficiente si todo el archivo está en memoria principal.
- A nivel físico se almacena como una lista enlazada de bloques.
- Al introducir nuevos registros tendría que mover los anteriores por de bloque en bloque para hacer huecos, lo que hacemos es usar una organización de tipo pila para los nuevos registros.

**b) Acceso Directo.**

El archivo está formado por registros lógicos de tamaño fijo para permitir que los programas lean

y escriban rápidamente y sin un orden en particular. Este método se basa en el modelo de archivo de disco; para el acceso directo se ve al archivo como una secuencia numerada de bloques o registros.

El acceso directo es de gran utilidad para el manejo inmediato de grandes cantidades de información y las bases de datos frecuentemente lo utilizan. Cuando llega una consulta sobre un sujeto en particular, primero se procesa aquellos bloques que contienen la respuesta, y luego se la arma en un bloque que se lee directamente, brindando la información deseada.

Un bloque es generalmente una cantidad de longitud fija, definida por el S.O. como la unidad mínima de posicionamiento. Puede ser 1 byte, 512 palabras, 1024 bytes, etc.

Algunos sistemas permiten definir distintos tamaños de bloques para cada archivo. El acceso directo permite leer o escribir bloques en forma arbitraria. Así se puede leer el bloque 14 y luego el 53 y luego escribir el 7. No existe ningún tipo de restricción.

Es de gran uso para acceso inmediato a grandes cantidades de información. Cuando se realiza una consulta, se computa el bloque que contiene la respuesta y se lee directamente el mismo.

Las operaciones de archivos deben modificarse para incluir el número de bloque como un parámetro. Así si se debe leer  $n$  en vez de leer el próximo, luego el siguiente, y luego el siguiente así sucesivamente hasta encontrar a  $n$ , como en acceso secuencial. El método debe permitir posicionarse en  $n$  en forma directa.

El número de bloque es normalmente un número relativo al comienzo del archivo. Así el primer bloque del archivo es el 0, luego el 1, etc., aun cuando el valor absoluto de bloque dentro del disco sea el 15703, el segundo el 15704, etc. Esto permite al S.O. decidir donde será ubicado el archivo (asignación o ubicación visto en el punto 7.4) y ayuda a evitar que el usuario acceda a porciones del sistema de archivos que no pertenezcan a su archivo. Algunos sistemas comienzan a numerar los bloques relativos con 0 y otros con 1.

Algunos sistemas requieren que se defina al archivo como secuencial o directo cuando es creado; así el acceso debe ser consistente con lo declarado en la creación. Notemos, sin embargo, que es fácil simular un acceso secuencial en un archivo con acceso directo. Contrariamente, es extremadamente ineficiente e incomodo simular un acceso directo sobre un archivo secuencial.

Decíamos que las operaciones con archivos deben recibir como parámetro el número de bloque. Con ello se realiza la operación *read  $n$* , donde  $n$  es el número de bloque, en lugar de *read next*; y *write  $n$*  en lugar de *write next*. Un enfoque alternativo para mantener *read next* y *write next*, es agregar la operación *posicionarse en  $n$* , donde  $n$  es el número de bloque y a partir de allí hacer un *read next* o *write next*.

El usuario normalmente proporciona al S.O. un *número de bloque relativo*. Un número relativo de bloque es un índice relativo al principio de archivo, su uso permite al sistema operativo decidir donde alojar el archivo y evita el acceso incorrecto al sistema de archivo. Algunos sistemas comienzan los números de bloque relativos en 0 y otros en 1.

Dado un tamaño de registro lógico  $L$ , una consulta al registro  $N$  se transforma en un pedido de I/O para  $L$  bytes desde la locación  $L + (N - 1)$  en el archivo. Como los registros lógicos son de tamaño fijo, es fácil leer, escribir o borrar un registro.

No todos los sistemas operativos soportan métodos de acceso secuencial y directo; decíamos que algunos sistemas obligan a definir el método de acceso en el momento de la creación del archivo.

ACCESO SECUENCIAL	IMPLEMENTACION EN ACCESO DIRECTO
reset	cp:= 0;
read next	read cp;
	cp:= cp+1;
write next	write cp;
	cp:= cp+1;

**Tabla 7.2** operaciones en acceso secuencial y directo

### c) Otros Métodos de Acceso.

Sobre el método de acceso directo se pueden construir otros métodos, generalmente incluyendo un *índice* al archivo. Esto se conoce como Acceso indexado que puede tener tres variantes: Secuencial indexado, Indexado Puro y Método Hash.

El índice, como en un libro, contiene punteros a los distintos bloques. Para encontrar una entrada al archivo, primero se busca en el índice y luego se usa el puntero para acceder directamente al archivo y encontrar la información deseada. Por ejemplo, un archivo de artículos puede tener una lista de Códigos de Productos Universal para ítems y sus precios asociados. Cada entrada consiste en un código consta de 10 dígitos y el precio de 6 dígitos (16 Bytes en total). Si el disco tiene bloques de 1024 Bytes, se pueden almacenar 64 registros por bloque. Un archivo de 120.000 registros ocuparía unos 2.000 bloques aproximadamente (2 MeBy). Manteniendo el archivo clasificado por código de

producto, podemos definir un índice consistente en el primer código. El índice tendrá 2.000 registros de 10 dígitos c/u, o 20.000 bytes, el que puede guardarse en memoria. Para encontrar un registro, buscamos -en forma binaria- en el índice y luego en el bloque exacto que contiene la información. Esto nos permite búsquedas en archivos muy grandes y con poca E/S.

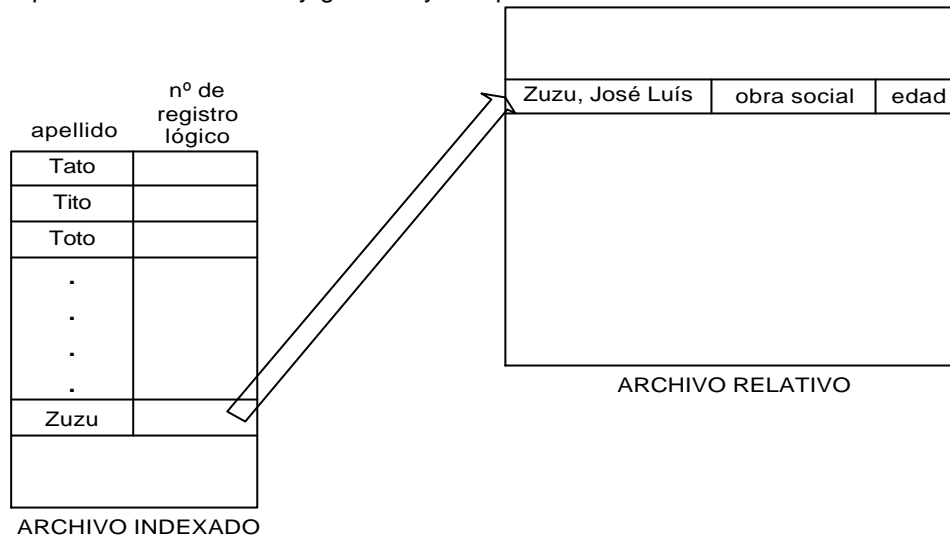


Fig 7.20 Archivos en el sistema VMS

Para archivos muy grandes hasta el índice se vuelve grande como para mantenerlo en memoria, en estos casos la solución es crear un índice para el archivo índice. El índice primario contiene punteros hacia el índice secundario, los que a su vez apuntan a los ítems de datos. Un ejemplo es el acceso ISAM (Index Sequential Access Method) que usa un pequeño índice maestro que apunta a bloques de un índice secundario. El archivo se encuentra clasificado por una clave definida.

Para encontrar un ítem en particular, se realiza una búsqueda binaria en el índice maestro, el que provee el número de bloque del índice secundario y el archivo está ordenado por una clave. Se realizan dos búsquedas binarias hasta encontrar el bloque deseado, el cual es recorrido secuencialmente. Este bloque es leído en forma idéntica al anterior y provee el número de bloque del registro deseado. Finalmente, se realiza una búsqueda secuencial en ese bloque.

Se recomienda para archivos demasiado grandes, crear un índice primario al índice secundario del archivo.

Por otra parte el sistema VMS realiza una implementación similar a la anterior; pero con archivos relativos.

#### Resumen sobre los tres métodos:

1. **Secuencial indexado:**
  - Es la solución al secuencial anteriormente explicado.
  - Se utilizan dos archivos auxiliares, un índice y uno de desbordamiento (overflow).
    - Índice: Permite hacer búsquedas para llegar cerca del registro buscado
    - OVF: Permite almacenar los registros que llegan nuevos.
  - Para buscar un determinado registro se busca en el índice el ínfimo de la clave, luego se mira donde apunta el puntero y sino está en esa posición tiene que estar en una posición enlazada del archivo OVF.
  - Es más eficiente cuanto más se parecen el número de entradas del índice con el número de cada partición inducida en el archivo principal.
  - Al final del registro, tanto en el archivo principal como en el de desbordamiento se tiene un puntero que apunta al registro siguiente.
  - Se puede tener más de un nivel de indexación, esto sería algo así como tener un índice para el índice, se gana rapidez a costa de sacrificar un poco de espacio en el disco duro.
2. **Puramente indexados:**
  - Se usan para efectuar búsquedas por varios campos, esto supone que no están ordenados y que no existe un campo clave.
  - Se tiene un índice por cada campo objeto de búsqueda.
  - No existe nada de secuencialidad.
  - Existen dos tipos de índices:
    - Exhaustivos: Existe una entrada por cada registro del archivo principal.
    - Parcial: Existen entradas para los registros donde se encuentra el campo buscado.
3. **Organización de tipo Hash:**
  - Se aplica una función o una tabla Hash al campo clave que estamos tratando de localizar.
  - Se usa sólo un archivo principal y uno de Desbordamiento.
  - Se suministra un campo clave a la función que devuelve directamente a una posición.

- La función determina la capacidad del archivo principal, en caso de que un registro fuese a entrar en una posición ya ocupada se le manda al archivo auxiliar.
- Se usa cuando necesito accesos muy rápidos de longitud fija.

## 7.6.- Operación sobre Archivos.

### 1. Operaciones sobre archivos como un todo

Un archivo es una forma de almacenar información para su posterior uso.

Para poder usarlo necesitamos realizar las siguientes actividades:

- **Abrir y cerrar** un archivo ya creado para poder usarlo.
- **crear** un archivo
- **escribir** el mismo.
- **rebobinar** el archivo - en caso de cinta - o reposicionar la cabeza en el caso de disco
- **leer** el mismo. Eventualmente, luego, si no nos sirve, podremos necesitar
- **borrar** el archivo. Estas son las siete operaciones mínimas requeridas para el manejo de archivos.

También se necesita:

- **Copiar:** podemos también crear copias de un archivo en el mismo o en otro dispositivo de E/S, tal como una pantalla o impresora.
- **Renombrar:** ya que los archivos son objetos nominados, muchas veces necesitaremos renombrar un archivo, es decir, darle otro nombre.
- **Editar el archivo y modificar su contenido.** Una modificación común es agregar nueva información al final del archivo ya existente.
- **Moverlo**(de un directorio o volumen a otro)
- **Buscarlo**(por su nombre) en todo el sistema de archivos

### 2. Operaciones sobre los registros

- **Recuperar Todo ( Retrieve\_all):** Recuperar todos los registros de un archivo. Esto va a requerir de una aplicación que deba procesar toda la información de un archivo una vez. Esta opción es usualmente equivalente con el termino de sequential proccessing, (proceso secuencial), porque todos los registros son accedidos en secuencia.
- **Recuperar\_Uno (Retrieve\_One):** Esta operación requiere la recuperación de un solo un registro. Las soluciones interactivas orientadas a la transacción necesitan esta operación.
- **Recuperar\_siguiente (Retrieve\_Next):** Esta operación implica la recuperación del registro que es el siguiente, según una secuencia lógica, el recuperado hace menos tiempo. Ej: aplicaciones interactivas como el llenado de formularios o un programa que efectúa búsquedas dinámicas.
- **Recuperar Previo (Retrieve\_Previous):** Es similar a Recuperar Siguiente, pero en este caso el registro que es "previo" al que sé esta accediendo en el momento actual. Trata con respecto al registro logicamente "anterior al actual" en algún determinado orden.
- **Insertar Uno (Insert One):** Inserta un nuevo registro dentro del archivo. Es necesario que el nuevo registro se ajuste a una posición particular para preservar la secuencia del archivo. En general es necesario que el registro insertado conserve el orden lógico de todo archivo. Si el archivo esta en disco debería tener lugar previamente reservado (si los registros son de longitud fija). En cintas se debe tener una de Entrada y otra distinta de Salida; a menos que sea APPEND (agregado al final del archivo anterior, alterando el orden lógico del archivo).
- **Borrar uno (Delete One):** Borra un registro existente. Ciertos enlaces o otras estructuras puede que necesiten actualizarse para preservar la secuencia del archivo. A veces se debe actualizar encadenamientos u otras estructuras de datos (ej: archivo de índices) a fin de conservar el orden lógico de un archivo (o secuenciamiento).
- **Actualizar un registro (Update\_one):** Recuperar un registro; actualizar uno o mas de sus campos y reescribir el registro actualizado en memoria central, manteniendo el orden lógico (no puedo actualizar una clave principal, para lograrlo se debe BORRAR el registro con la vieja clave y AGREGAR el registro con la nueva clave). Si la longitud del registro actualizado ha variado (demasiado si la longitud creció), implica que su reescritura puede resultar muy dificultosa. No es así si la longitud de los registros es siempre FIJA. Es necesario preservar la secuencia con esta operación. Si el tamaño del registro esta cambiado, la operación de actualización es más difícil si el tamaño es preservado. Luego se graba el archivo
- **Recuperar Varios (Retrieve\_Few):** Recupero un número de registros. Por ej: recuperar solo los registros que cumplan un determinado criterio.

- **Append:** anexar

**LA NATURALEZA** de las operaciones que se deban efectuar más frecuentemente sobre los registros de un archivo **DETERMINAN SU ORGANIZACIÓN**. Las organizaciones pueden ser: secuencial, secuencial indexada o directa y el tipo de acceso: secuencial, indexado al azar o dinámico, y al azar.

El concepto de archivos es implementado por el S.O. Este provee llamadas al sistema (SYSCALL) para abrir, cerrar, crear, escribir, leer, rebobinar y borrar archivos.

Para entender como son soportados los sistemas de archivos, veremos más en detalle las siete operaciones básicas sobre archivos, donde se observará la importancia del SYSCALL respectivo y en particular el directorio de dispositivo.

Estas operaciones básicas conforman el juego indispensable de operaciones para el manejo de archivos. Las operaciones primitivas pueden ser combinadas o concatenadas y conformar nuevas operaciones. Por ejemplo un **copy** de un archivo, primero se crea un nuevo archivo y luego se lee el viejo y se escribe el nuevo.

También deben existir operaciones que permitan al usuario **consultar** los atributos del archivo (por ej. su tamaño) y operaciones que permitan al dueño del archivo manipular sus atributos.

La mayoría de las operaciones mencionadas involucran la búsqueda del registro en el directorio asociado al nombre del archivo; para evitar esta búsqueda constante algunos sistemas guardan una pequeña tabla con información sobre todos los archivos abiertos (la *tabla de archivos abiertos*), otros utilizan en el PCB del proceso un área para los archivos abiertos de ese proceso. Una vez que un archivo se involucra en alguna operación se lo incluye en la tabla, cuando se necesita una nueva operación se usa un índice hacia la tabla y de ésta manera se evita la búsqueda en el directorio. Después de un determinado tiempo de inactividad, el sistema puede cerrar, o no, el archivo y borrar la entrada en la tabla.

Algunos sistemas implícitamente abren un archivo cuando se los referencia por vez primera y lo cierran automáticamente si el programa termina su ejecución. Pero la mayoría exige que el programador abra explícitamente el archivo - con una llamada al sistema - antes de usarlo.

Asumiremos que el sistema de archivos esta basado en discos. Observemos que debe realizar el S.O. para implementar las operaciones básicas.

### 7.6.1. Operación de Apertura y Cierre de un Archivo.

Es una de las operaciones más importante del Sistema de Gestión de Archivos, ya que se ocupa de preparar los accesos en forma segura a la estructura de datos almacenada en el soporte. Digamos que en memoria central se está procesando en el área del usuario la instrucción:

**open** (Nombre\_archivo = toto, modo de acceso = Leer), :\* Puntero\_vinculación;

de un Proceso que necesita abrir un archivo (Ejemplo "TOTO") y que genera la llamada al sistema para que le provea el servicio mediante el siguiente SYSCALL:

**open** (Nombre\_archivo = toto, modo de acceso = Leer),

y devuelve ".\* Puntero\_vinculación" que apunta al FCB ;. Esta llamada genera las acciones indicadas en el esquema de la Fig. 7.21. Entonces se comienza a ejecutar la función de la figura 7.22 que inicia una búsqueda en el directorio (VTOC) para encontrar la entrada del Archivo TOTO con las actividades detalladas en la Fig. 7.22, y luego genera una estructura de datos que denominaremos genéricamente **File Control Block (FCB)** que presentamos en la Fig. 7.23. Después produce las vinculaciones (**binding**) con el módulo de Entrada/Salida (Bloque de Control del Driver -BCD - Bloque de Control de la Unidad -BCU- ambos creadas durante el proceso de inicialización del Sistema - Booteo-), y el VTOC del dispositivo (creado durante el formateo del soporte y actualizado durante las operaciones sobre archivos). Hecho todo esto queda habilitada la vinculación para el uso del archivo TOTO (Ver los detalles de los contenidos de las tablas en la Fig. 7.34).

Si bien habíamos mencionado que algunos sistemas abren un archivo en su primer referencia y lo cierran cuando el proceso que lo abrió termina; la mayoría requieren que el programador abra el archivo con un syscall (**open**) para luego usarlo. La operación open toma el nombre del archivo y lo busca en el directorio, copiando la entrada asociada en la tabla de archivos abiertos; luego retorna un puntero a la tabla el que se usará para las operaciones de Entrada - Salida. En la Figura 7.21 se describe la vinculación que realiza el **open ()**.

La implementación de las operaciones **open()** y **close()** en un ambiente multiusuario, como ser UNIX, es bastante complicada ya que varios usuarios podrían abrir el archivo al mismo tiempo. Normalmente, el S.O. utiliza dos niveles de tablas internas; una tabla por proceso (lo indicamos dentro del PCB) conteniendo todos los archivos abiertos. En ésta tabla se almacena información concerniente al uso del archivo por parte del proceso, por ejemplo el *puntero de posición actual* en el archivo (**Current File Pointer**).



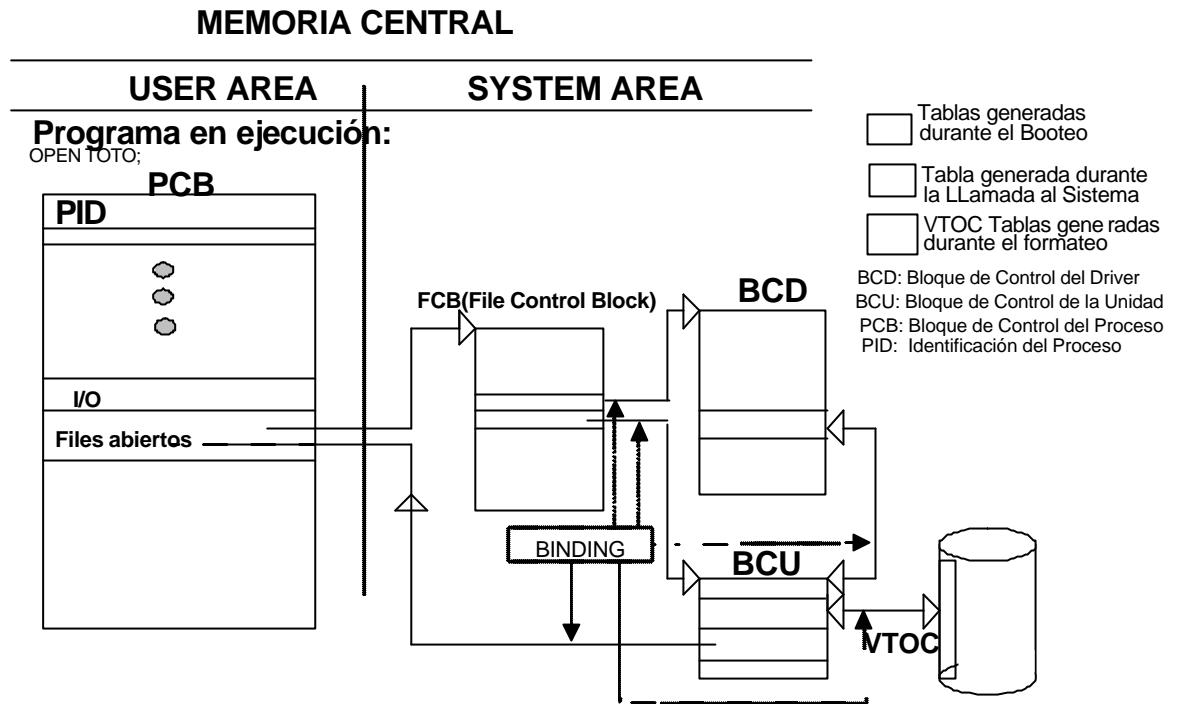


Fig. 7.21 Vinculación de Tablas realizadas por la operación open

Cada entrada de la tabla por procesos apunta a una tabla general de archivos abiertos (una por archivo - FCB); ésta tabla contiene información independiente al proceso, como ser su ubicación en el disco, datos de acceso o tamaño.

(\*) En realidad la operación de *buscar en el directorio* como cualquier otro acceso requiere de un conjunto de actividades (A, B, C y D) que son esquematizadas en la [figura 7.23](#)

Cuando un proceso ejecuta un open de un archivo que ya está abierto por otro, el resultado es una nueva entrada a la tabla FCB del archivo del proceso, con un nuevo puntero de posición actual y el correspondiente puntero a la tabla general conteniendo todos los FCB. Normalmente en la tabla general existe un contador de aperturas realizadas sobre cada archivo.

Cada cierre (*close*) ejecutado decrementa el contador, que al llegar a cero indica que el archivo no está más en uso; por lo que se remueve su entrada de la tabla general.

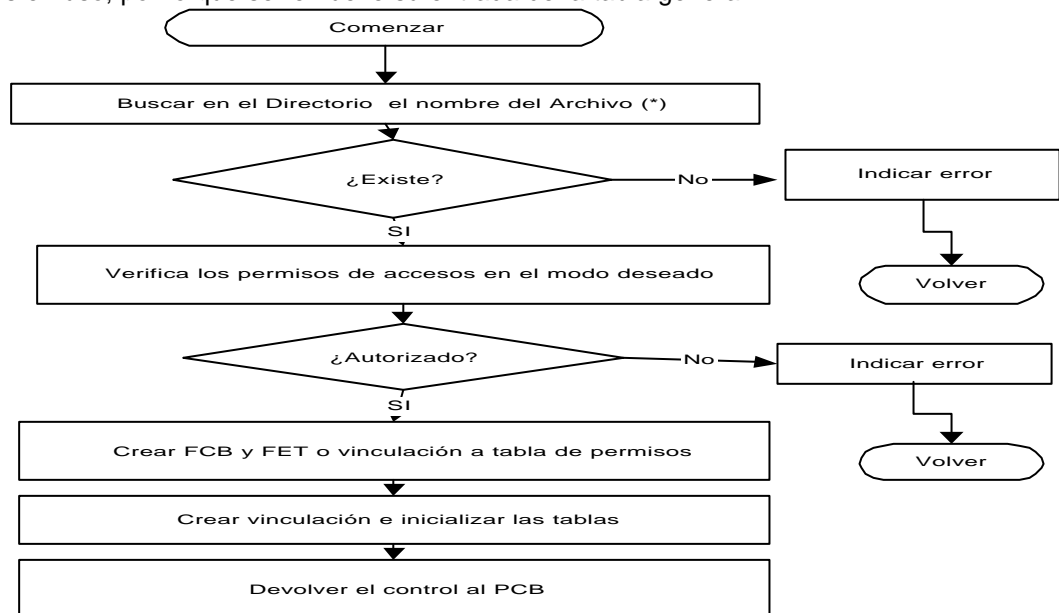


Figura 7.22 El algoritmo para la apertura de un archivo

Resumiendo: se necesita diferente información sobre un archivo abierto como ser:

- ❖ **PUNTERO AL ARCHIVO:** En los sistemas que no incluyen el desplazamiento de archivo (*offset*) como parte de las llamadas al sistema *read* y *write*, se debe rastrear la última



ubicación que fue leída/escrita con el puntero de posición actual (CFP), este puntero es único para cada proceso.

- ❖ **CONTADOR DE APERTURA DE ARCHIVOS:** Cuando los archivos se cierran el S.O. debe reutilizar sus entradas para no agotar el espacio de la tabla. Si varios procesos pueden abrir el mismo archivo, el sistema debe esperar a que todos los procesos cierren al archivo para borrar su entrada en la tabla de archivos abiertos. Este contador rastrea el número de opens y closes. Cuando se vuelve cero el sistema puede borrar la entrada.
- ❖ **UBICACIÓN EN EL DISCO:** La mayoría de operaciones con archivos necesitan modificar datos dentro de ellos. La información necesaria para ubicar al archivo en el disco se mantiene en memoria para permitir su consulta en cada operación.

Algunos sistemas operativos (MULTICS, fue el primero) permiten bloquear sectores de un archivo abierto para acceder con multiprocesamiento, compartir sectores entre varios procesos y hasta para "mapear" sectores de un archivo en memoria convencional o virtual. La última se denomina *mapeado de archivos a memoria*, y permite asociar lógicamente un espacio de direccionamiento virtual con un sector de un archivo.

Esta vinculación a través del FCB permite compartir los archivos entre varios usuarios dado que cada PCB tendrá el puntero al FCB y éste dispondrá una entrada con los permisos de acceso para cada usuario.

Este Bloque de control del Archivo (FCB) reside en Memoria Central en el espacio de direccionamiento privado del S.O. sin que el usuario pueda accederlo.

Nombre Simbólico del Archivo
Localización en el almacenamiento : ( tres Punteros : ComienzoBOF, CFP y Fin (EOF))
Organización: ( Secuencial, Secuencial-indexado, Directo, Indexado, etc.)
Volumen : (Disco = Unidad, Cinta = N° Tape) puede contener Vol. actual, último, etc.
Datos para el Control de ACCESO (uno por usuario)
Tipo de Archivo - Extensión- Versión (bin, obj, exe, dat, ASCII, backup, etc.)
Tratamiento (Permanente, Temporal, Inicialización, oculto, etc.)
FECHAS y horas: (creación, última modificación, último acceso, destrucción, etc.)
Prioridades
Conjunto de Punteros a programas de permisos, matriz de permisos, o flags para la operación ( Oculto, solo lectura, escritura o ejecución, Sistema, cerraduras, etc.)
Tamaños : (actual, máximo, etc.)
Área de Punteros a PCB, BCD, BCU, tabla de comandos, etc.)
Sección de Control de bloques (tamaño y N° del bloque actual, área de trabajo, buffers, etc)
Sección de Control de Registros (longitud y N° de registro por bloque, definición de Registros, dirección del registro actual, N° de Bytes que quedan en el bloque actual, flags varios, área de trabajo, salidas de usuario (por ejemplo EOF), etc)

Fig. 7.23 FILE CONTROL BLOCK (FCB)

### 7.6.2. Creación (Create) de un Archivo:

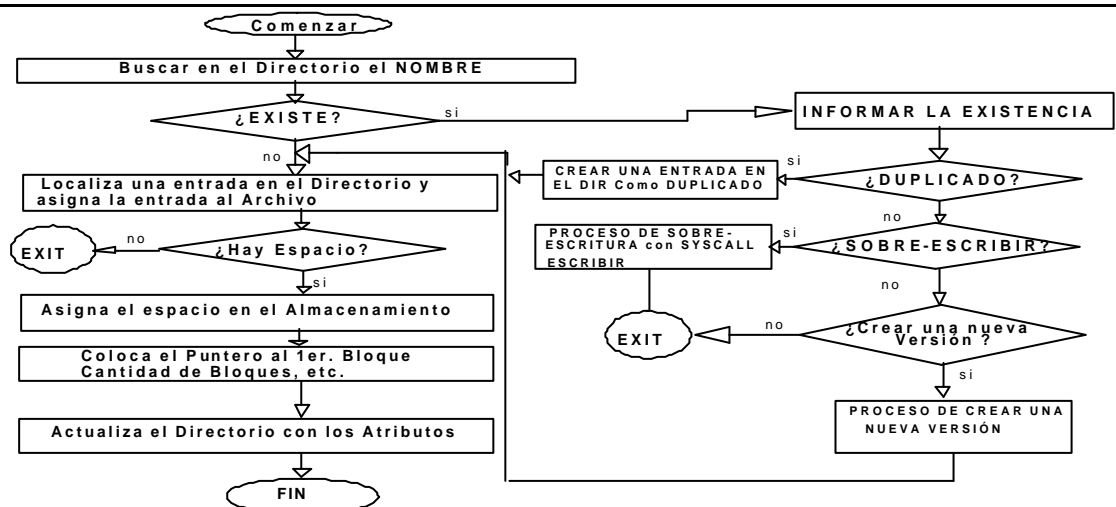


Fig. 7.24 Llamada al Sistema para Crear un Archivo

Se necesitan básicamente dos pasos para crear un archivo.

Primero, debe encontrarse suficiente espacio libre en el sistema de archivos para el mismo.

Mas adelante en este módulo discutimos las formas de asignación de espacio libre

Segundo, se debe crear una entrada en el directorio, es decir se graba el nombre en la entrada del mismo y su locación en el sistema de archivos y sus atributos.

La función será: **crear** (nombre\_archivo, atributos);

y el diagrama lógico sería el de la **figura 7.24**.

### 7.6.3. Escritura (Write) de un Archivo

Se realiza mediante una llamada al sistema en que se especifica el nombre del archivo y la información que se desea escribir sobre el mismo, la cantidad de bytes y el buffer de salida:

**escribir** (\* Puntero\_vinculación o Nombre\_archivo, cant\_bytes, buffer\_salida) : estado;

y el respectivo SYSCALL será:

SYSCALL estado = **escribir** (:\* Puntero\_vinculación o Nombre\_archivo, cant\_bytes, buffer\_salida);

que se esquematiza en la **fig. 7.25**

Con el nombre, el S.O. busca en el directorio para ubicar la locación del mismo. En el directorio se tiene grabado tres punteros importantes:

- el puntero al comienzo del archivo (primer bloque asignado) (*Begin of File Pointer - BOFP*),
- un puntero a la posición actual donde está posicionado (*Current File Pointer CFP*) y
- un puntero con el fin-de-archivo (*End Of File pointer EOF*).

Usando este último puntero, la dirección del próximo bloque puede computarse y así puede escribirse la información a continuación o se posiciona el CFP en el sector que se desea escribir y se ordena la escritura.

El sistema verificará que el archivo este abierto en el modo escribir, las protecciones, si esta en uso compartido y brindará los servicios necesarios. Si todo esta bien, devolverá el estado en la palabra de estado (Status Word).

Generalmente, para posicionarse en un archivo que está siendo escrito o leído, la mayoría de los sistemas utiliza un solo puntero, que indica la posición actual en que se está dentro del archivo. Esto reduce la complejidad del sistema y economiza lugar en el directorio.

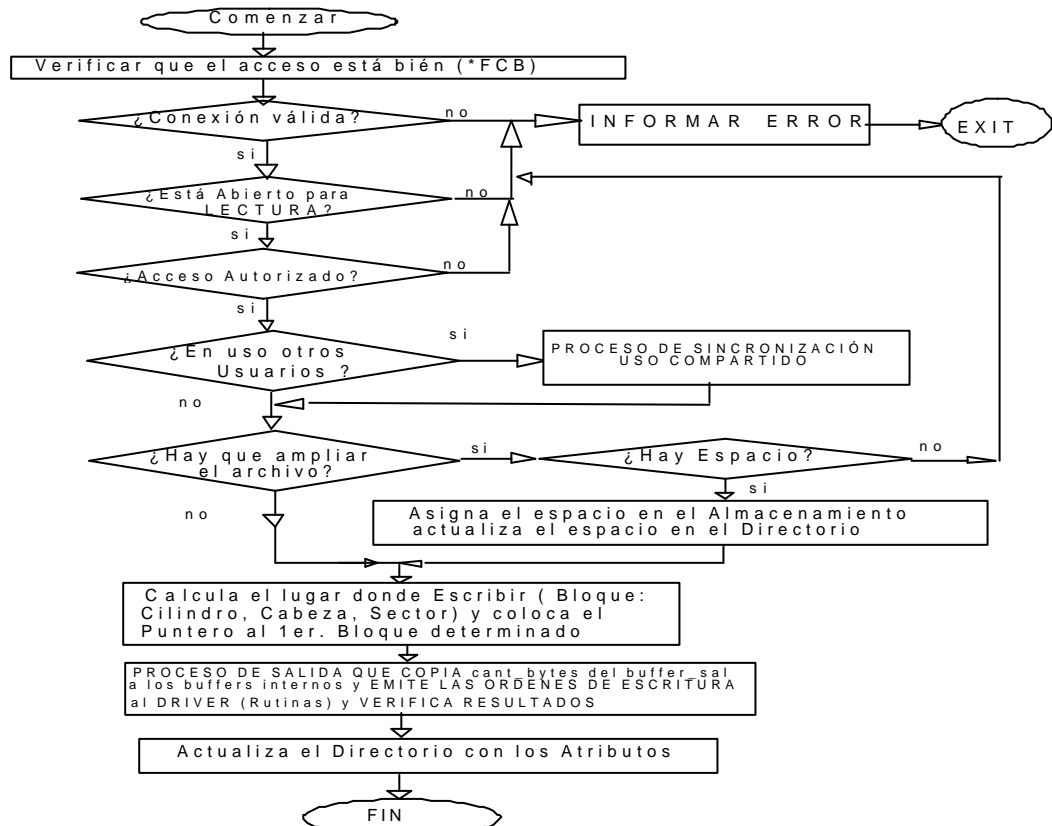


Fig. 7.25 Llamada al Sistema para Escribir en un Archivo

#### 7.6.4. Lectura (Read) de un Archivo:

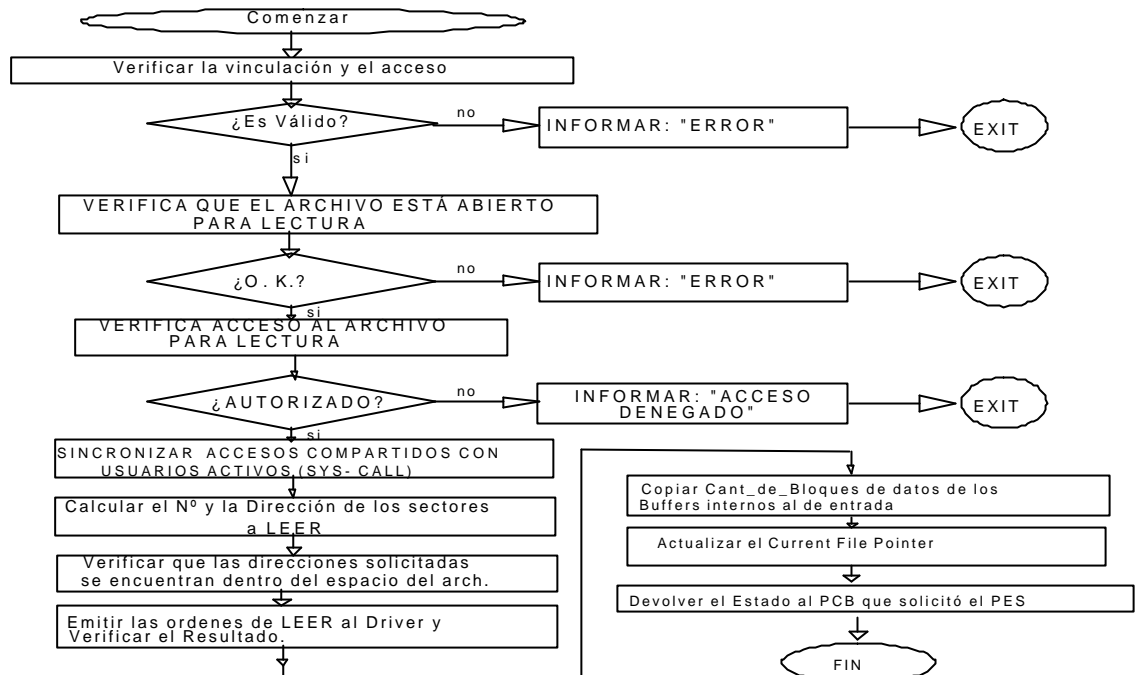


FIG. 7.26 Llamada al Sistema para una operación de Lectura en un Archivo

Se realiza mediante una llamada al sistema donde se debe especificar el nombre del archivo a leer, el puntero de vinculación y en que lugar de la Memoria Central se colocará el siguiente bloque leído. La función sería:

leer (:\* Puntero\_vinculación o Nombre\_archivo, cant\_bytes, buffer\_entrada) : estado;

y el respectivo SYSCALL será:

SYSCALL estado = leer (\* Puntero\_vinculación o Nombre\_archivo, cant\_bytes, buffer\_entrada);

que se esquematiza en la **fig. 7.26**.

Nuevamente se accede al directorio para localizar el archivo y también se necesita el puntero CFP al próximo bloque de lectura. Una vez leído, se actualiza el puntero.

#### 7.6.5. Rebobinado (Rewind) de un Archivo:

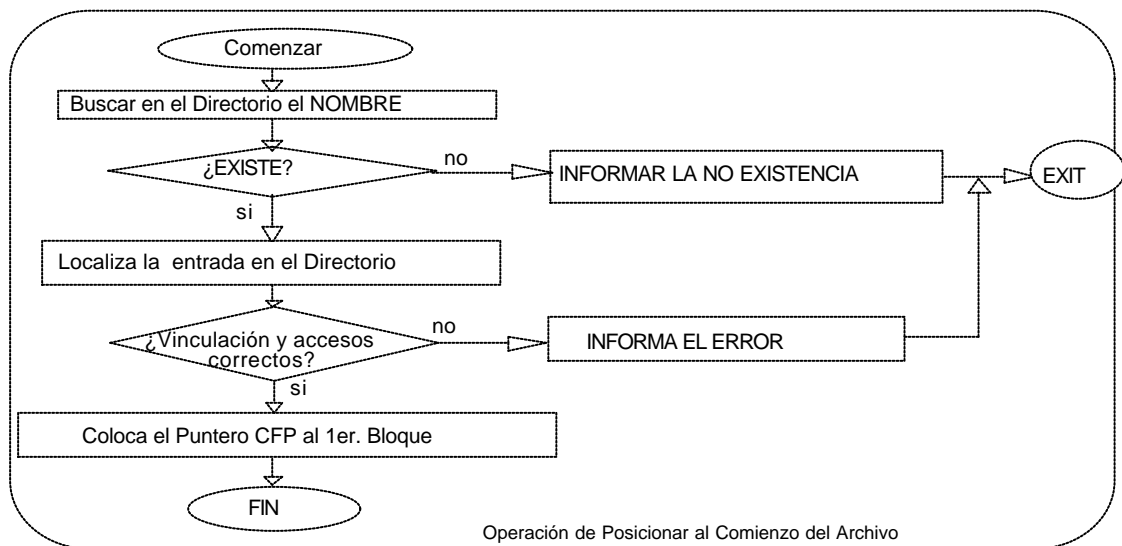


Fig. 7.27. Operación de posicionar al comienzo del archivo

apunta el current file pointer al principio del archivo. Entonces la operación consiste en acceder al directorio para ubicar el archivo y se reposiciona el puntero al comienzo del archivo (primer Bloque).

#### 7.6.6. Borrado (Delete) de un Archivo:

Para borrar un archivo que se desea eliminar, se lo busca en el directorio. Una vez encontrada la

entrada del archivo, se libera todo el espacio ocupado por el mismo (así puede reasignarse a otros archivos) y se invalida la entrada al directorio.

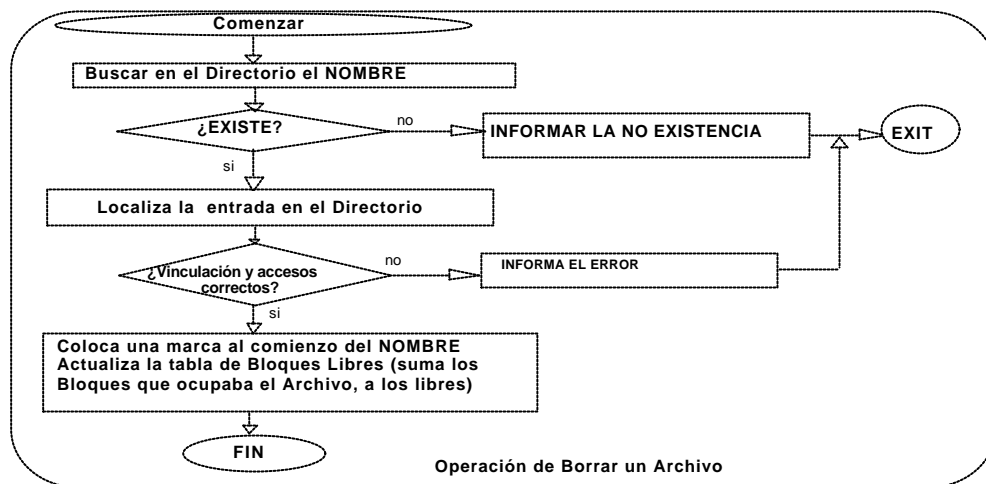


Fig. 7.28. Operación de borrado del archivo

## 7.7. Sistemas de Directorio

El directorio de archivos está asociado con:

- 1) una colección de archivos y
- 2) cualquier sistema de administración de archivos.

El directorio contiene información sobre los archivos, incluyendo atributos, ubicación y propietario.

La mayoría de la información contenida en el directorio (especialmente aquella relacionada con el almacenamiento) es administrada por el sistema operativo.

El directorio es un archivo cuyo dueño (propietario) es el S.O., accesible a través de diversas rutinas de gestión de archivos.

La información contenida en los directorios sólo es accesible por los usuarios (y los programas) provistas indirectamente por rutinas del sistema.

Los usuarios **NO PUEDEN** acceder al directorio **DIRECTAMENTE**, ni siquiera a modo solo lectura (read only). Para cada archivo del sistema la información típicamente almacenada en el directorio es la siguiente:

**INFORMACIÓN BÁSICA:** (nombre del archivo, tipo de archivo (texto binario, módulo de carga, etc.), organización),

**INFORMACIÓN DE DIRECCIONAMIENTO:** (volumen (ubicación física del soporte donde se encuentra el archivo), dirección física de comienzo del archivo; cantidad usada de almacenamiento (longitud actual del archivo), cantidad asignada (longitud máxima del archivo),

**INFORMACIÓN DE CONTROL DE ACCESOS:** (propietario del archivo (quien autoriza su uso por parte de otros usuarios y quien revoca esos permisos), información de acceso (nombre y clave para cada usuario autorizado) y tipo de acceso permitido (read, write, execute, transmitir por red)).

**INFORMACIÓN DE USO:** (fecha de creación del archivo, identificación del creador, fecha del último acceso de lectura e identificador del lector, fecha de la última modificación e identificación de quién efectuó la modificación; fecha del último resguardo (backup) y uso actual (información acerca de la actividad actual del archivo) como ser: proceso (o procesos) que tiene/n el archivo abierto, si está o no bloqueado (locked) por un proceso y si el archivo fue modificado en memoria pero no en el disco todavía.

Desde el punto de vista del usuario el directorio provee un mapeo entre los nombres de archivos (conocidos por los usuarios y las aplicaciones) y los archivos en sí mismos, por eso cada entrada de directorio incluye el nombre del archivo. Virtualmente todos los sistemas operan con distintos tipos de archivos (al menos el tipo "ejecutable" o módulo de carga (load module)) y distintas organizaciones de archivos.

En los sistemas compartidos es importante proveer información con relación al control de accesos al archivo. El usuario "dueño" del archivo es quien otorga y quita los permisos de acceso.

Finalmente se necesita información de uso del archivo, tanto actual como histórica. La información actual se necesita en memoria mientras el archivo está en uso ("abierto").

A medida que la cantidad de usuarios y de información almacenada crece, puede ser difícil

para los usuarios poder localizar los archivos con la estructura vista de directorio de dispositivo. La solución es una estructura de directorios para el sistema. Esto provee un mecanismo para la organización de muchos archivos y puede exceder los límites de una unidad de disco e incluir a varias. Así al usuario solo le interesa la estructura de archivos y de directorios lógicos, pudiéndose desentender del espacio de archivos por unidad.

Muchas y distintas estructuras se han propuesto y están en uso.

### 7.7.1. Directorio de Dispositivo

La información específica de cada archivo es guardada en una lista de símbolos llamada directorio que varía de sistema en sistema. En general podemos resumir en una tabla en el disco llamada *File Directory Block* que contiene algunos de los siguientes ítems:

Información de encabezamiento:

- Nombre del directorio
- Puntero a la lista del Directorio (Master File Directory) y los Subdirectorios (User File Directory) que contiene.

Cada Directorio del Usuario tiene la siguiente información por cada entrada o archivo como se indica en la **Figura 7.29**

### 7.7.2. Operaciones Sobre Directorios.

A continuación examinaremos distintas estructuras de directorio de archivos. Al estudiar una estructura particular debemos tener en cuenta las operaciones que se pueden realizar sobre un directorio:

<b>Información básica por cada entrada (entry) en la tabla de símbolos:</b>
<b>Nombre de archivo: nombre simbólico (generado por el creador. Único en el Directorio)</b>
<b>Tipo o extensión de archivo: (Para aquellos sistemas que soportan distintos tipos)</b>
<b>Organización (para aquellos File System que soportan distintas organizaciones)</b>
<b>Información de dirección:</b>
<b>Volumen: para el Dispositivo que contiene el archivo</b>
<b>Ubicación: Un puntero al dispositivo y ubicación del comienzo en ese dispositivo del archivo-BOF Begin Of File- (Cilindro, superficie o cabeza y sector).</b>
<b>Puntero de la ultima posición del Archivo (End Of File pointer)</b>
<b>Posición actual: Un puntero a la posición actual de lectura o grabación (Current File Pointer)</b>
<b>Tamaño: El tamaño actual del archivo (en bytes, palabras o bloques) y el máximo permitido.</b>
<b>Información de Control de Acceso:</b>
<b>Dueño: puede fijar los privilegios de accesos (permisos).</b>
<b>Protección: Información de control de acceso para lectura, escritura, ejecución, etc. por cada usuario autorizado.</b>
<b>Contador de uso: Indica el número de procesos que estan usando (es decir que han abierto el archivo).</b>
<b>Información de uso:</b>
<b>Hora, día e identificación de proceso: Esta informacion puede ser de 1) creación, 2) última modificación, 3) último uso o acceso o 4) último Back-up. Esto puede ser útil para protección y monitoreo de uso</b>
<b>Etc.</b>

Fig 7.29 Información por cada Entrada de un archivo en el directorio

**Crear un directorio:** Se crea un directorio vacío, (en el MS-DOS excepto por "punto" y "punto-punto" que se colocan en forma automática por el sistema (o en algunos casos por el programa, por ej. *Make directory o md*) en el caso del MS-DOS). "Punto" y "punto-punto" ( "." y "..") son entradas especiales de todo directorio y tienen relación con el sistema jerárquico de directorios soportado por dicho S.O. Punto indica el directorio de trabajo; punto-punto indica su padre. Para analizar su funcionamiento, veremos más adelante las estructuras de árbol de directorios.

**Buscar (SEARCH):** un archivo Si se necesita buscar la entrada de un archivo particular, se debe explorar la estructura de directorios para encontrar la entrada de ese archivo. Se requiere estar habilitado (disponer permisos) para recorrer la estructura del directorio y encontrar un archivo en particular. Dado que los archivos tienen nombres simbólicos y nombres similares pueden indicar una relación entre ellos, a veces se requiere buscar por ejemplo todos los archivos que satisfacen determinado criterio o se debe poder encontrar todos los archivos cuyos nombres concuerdan con un patrón determinado. Por ejemplo tal vez quisiéramos encontrar todos los archivos con la particularidad común de que sean documentos.

**Eliminar o sea Borrar (DELETE) un archivo:** Cuando un archivo deja de ser necesario (no se necesita más) se lo puede eliminar del directorio. Se libera esa entrada.

**Crear (CREATE) un archivo:** Los archivos nuevos necesitan crearse y agregarse al directorio. Para ello se busca una entrada libre o se crea una entrada nueva y se graban los parámetros, el nombre lógico del archivo, su extensión, etc. y se preparan los punteros BOF, CFP y EOF. (ver [figura 7.24](#))

**Renombrar un archivo:** Como el nombre del archivo es para el usuario representativo del contenido, debe ser modificable cuando el contenido cambia o simplemente desea cambiar de nombre. La acción de renombrar debe permitir cambiar su posición en la estructura del directorio.

Recorrer el sistema de archivos: Es de gran utilidad poder acceder a todos los directorios y a todos los archivos de la estructura del directorio. Para mantener su integridad, es bueno grabar el sistema de archivos con su estructura y contenidos cada tanto a través de una copia de seguridad actualizada.

**Eliminar un directorio:** Para eliminar un directorio este debe estar vacío, (excepto en MS-DOS por punto y punto-punto, ya que generalmente no se pueden eliminar si no está vacío en algunas versiones). Algunos S.O. permiten eliminar todo el directorio con sus archivos. Entonces de acuerdo a la política que implemente el sistema operativo el borrado de los directorios se maneja de dos formas alternativas:

Se elimina el directorio con todos los archivos y subdirectorios que estén debajo.

Sólo se elimina el directorio si está vacío.

**Listar el directorio (LIST DIRECTORY):** Esta acción se hace para saber que archivos hay en el directorio y que extensiones posee cada uno, o que atributos dispone. Esta lista generalmente se presenta en pantalla con un formato predeterminado con los contenidos de cada entrada o registro de archivo que hay en él..

**Copias de seguridad (BACKUP):** Por razones de seguridad es aconsejable guardar periódicamente los contenidos y estructura del Sistema de Gestión de Archivos. Se suele llevar a cabo copiando todos los archivos a cinta magnética, o Discos lo cual provee una copia de seguridad (copia de resguardo) en caso de una falla del sistema o un daño en el soporte poder recuperar la información dañada. También se utiliza para guardar archivos fuera de uso y así su espacio puede ser ocupado por otros archivos.

La lista sencilla no es adecuada para soportar estas operaciones.

El usuario va a disponer de varios tipos de archivos (archivos de procesadores de texto archivos gráficos hojas de calculo archivos de sonido, etc.).

El usuario puede querer tener sus archivos organizados por tipo o por proyecto o por cualquier otro criterio. Un directorio tipo lista no provee de ayuda para organizar los archivos y fuerza al usuario a ser cuidadoso y no poner igual nombre a dos archivos de distinto tipo.

El problema aumenta en un sistema compartido: los nombres únicos constituyen un problema. Además es muy difícil poder ocultar porciones del directorio a dos usuarios cuando no hay una estructura inherente en el directorio. Para comenzar a resolver este problema se uso un esquema de dos niveles. Aquí hay un directorio maestro y luego un directorio para cada usuario. El directorio maestro tiene una entrada para cada directorio de usuario: para proveer espacio e informar el control hace eso. Cada directorio de usuario es una lista de los archivos de ese usuario. Este arreglo hace que la unicidad de nombres solo sea necesaria dentro de cada colección de archivos (los archivos pertenecientes a cada usuario); y ahora el file system puede forzar restricciones de acceso a nivel directorio de usuario. Aquí tampoco se ayuda al usuario a estructurar (jerárquicamente colecciones de archivos).

Una aproximación (a la solución de este problema) mas potente y flexible y casi universalmente usado, es la estructura jerárquica o arbórea. Como siempre hay un directorio maestro que tiene bajo si un conjunto de directorios de usuario. Cada uno de esos directorios de usuarios puede tener como entradas descripciones de archivos u otros subdirectorios, y esto es valido a todo nivel. En cualquier nivel un directorio puede consistir de entradas para subdirectorio y/o de entrada para archivos. Queda por decir como organizar cada directorio y subdirectorio. La aproximación mas sencilla es guardar cada directorio como un archivo secuencial. Cuando los directorios contienen un gran numero de entradas, esta organización puede organizar tiempo de búsquedas muy largo. En este caso es preferible una estructura directa (hashed)

### 7.7.3. Estructuras de Directorio

Lo que hemos visto hasta ahora nos permite crear archivos, leerlos, escribirlos y eliminarlos. Los archivos almacenados en el dispositivo se representan simbólicamente por entradas en el directorio o VTOC (*Volume Table Of Contents*: Tabla de contenido del dispositivo); cada una de estas entradas posee

el nombre simbólico, dirección, tamaño, fecha de último acceso, protección, etc. Debemos tener en cuenta que el nombre simbólico es lo que permite referenciar a un archivo y por lo tanto este nombre debe ser *único* en el sistema.

El directorio es finalmente una *tabla de símbolos*. El sistema operativo recibe el nombre simbólico de un archivo y lo encuentra en esta tabla.

Un directorio de dispositivo puede ser suficiente para un sistema monousuario con espacio de almacenamiento limitado. Pero a medida que el número de usuarios y la demanda de espacio crecen, será cada vez más difícil para los usuarios organizar y encontrar y manipular sus archivos.

La solución a este problema es dar al Sistema de Archivos una estructura de directorio. Una **estructura de directorio** provee un mecanismo para organizar la gran cantidad de archivos que posee el Sistema de Archivos, de forma tal que el usuario no necesite saber de que forma está asignado el espacio del soporte.

La mayoría de los sistemas operativos actualmente tienen dos estructuras de directorios separadas: el directorio de dispositivos también llamado master file directory y el directorio de archivos o user file directory.

1. **Directorio del dispositivo (device directory):** Este directorio se almacena en *cada* dispositivo físico y las entradas del mismo guarda los nombres de todos los archivos que contiene y describen las características físicas de cada archivo: Dirección de almacenamiento (donde está), longitud que tiene el archivo, forma de asignación de espacio (cómo está alocado), etc.
2. **Directorio de archivos (file directory):** Es una organización lógica de los archivos de *todos* los dispositivos y concentra información de propiedades lógicas. Cada entrada describe las características lógicas (extensiones) de cada archivo: Nombre, tipo, usuario propietario, información contable, información para auditoría, códigos de protección de acceso, etc. Una entrada apunta a la entrada del directorio de dispositivo para tener las propiedades físicas o simplemente duplican esta información.

Ambos Directorios lo describiremos como Master File Directory y User File Directory. Nuestro interés se centrará en la estructura de directorios de archivos que puede ser una sola tabla

### a). Estructura de Datos usada para los Directorios de Archivos.

Una lista lineal de entradas a un directorio requiere de una búsqueda lineal para encontrar una entrada en particular. Esto es simple de programar pero la ejecución demanda mucho tiempo. Para crear un nuevo archivo, se debe buscar en todo el directorio si no existe una entrada con igual nombre, luego agregar una nueva entrada al final del directorio. Para borrar un archivo, se busca en el directorio el archivo nominado, luego se libera la entrada con ese nombre. Para reusar el lugar de esa entrada se puede hacer de varias maneras. Algunos S.O. marcan esa entrada con una marca o carácter en particular (por ejemplo se llena con todos blancos) o se usa un bit especial de usado / no usado o se tiene una lista de entradas libres en el directorio. Otra alternativa es copiar la última entrada al directorio en el lugar vacío, disminuyendo la longitud del directorio. Pero la real desventaja de esta estructura es el tipo -lineal- de búsqueda de una entrada. Un ordenamiento clasificado, permite una búsqueda binaria y baja el tiempo promedio de búsqueda. Sin embargo, el algoritmo que se emplea para la búsqueda es algo más complicado de programar. Además, la lista siempre debe estar ordenada. Este requerimiento puede complicar la creación y borrado de archivos, ya que tenemos que mover una cantidad importante de información para mantener el directorio clasificado.

Otro tipo de estructura usada frecuentemente para un directorio es una tabla hash (porción), que mejora notablemente el tiempo de búsqueda. La inserción y borrado de archivos puede realizarse sin inconvenientes, aunque debe tenerse cuidado en el caso de colisiones - situaciones donde dos nombres de archivos pretenden apuntar a una misma dirección. La mayor dificultad con este esquema es que generalmente usa un tamaño fijo de tabla.

#### Resumen sobre directorios:

- Organización de archivos en una unidad de almacenamiento.
- Tabla de símbolos en los que se asocia un nombre simbólico con un archivo. También es un archivo que es gestionado completamente por el S.O.
- Permite establecer una jerarquía de archivos.
- La organización que se usa es secuencial pura y puede estar fragmentado. Si el número de archivos que gestiona es muy elevado se puede optar por una función Hash.

### b). Directorio de un solo nivel o de nivel único (Single Level Directory)

Esta estructura tiene muchas limitaciones, por ejemplo, cuando aumenta el número de archivos o cuando existe más de un usuario. Como todos los archivos pertenecen al mismo directorio deben tener nombres únicos (no repetidos), si dos usuarios quieren llamar a su archivo *TATO* están violando esta

regla. Los nombres de archivos generalmente se eligen reflejando su contenido, por lo tanto están limitados por el largo permitido.

Aún con un solo usuario, si aumenta el número de archivos y todos tienen un nombre único, es difícil recordar todos los nombres y rastrear un archivo se vuelve una tarea difícil.

Es la estructura más simple. Un ejemplo de esta estructura es el directorio del dispositivo. Todos los archivos están en el mismo directorio como se indica en la **figura 7.30**.

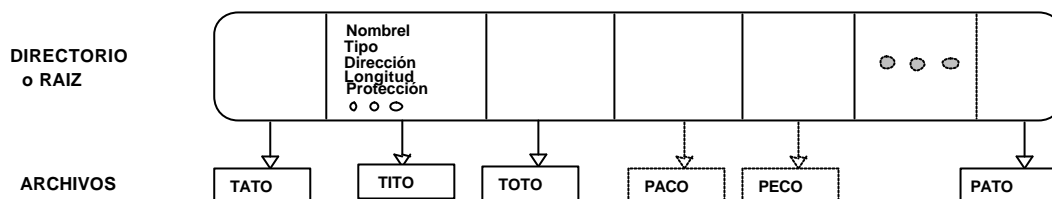


Figura 7.30 - Directorio de un solo nivel

**Ventajas:**

- Fácil implementación.
- Fácil de usar por el usuario.

**Desventajas:**

- Cuando crece el número de archivos o el número de usuarios la regla de unicidad de nombres simbólicos es cada vez más difícil de cumplir, y la búsqueda dentro del directorio se torna más lenta.
- No permite agrupar archivos con características comunes. Por ejemplo, sería deseable separar los archivos de datos de los archivos del sistema.
- La mayor desventaja de esta estructura es la confusión de los nombres de archivo entre diferentes usuarios. La solución standard es crear directorios separados para cada usuario. Especialmente en grandes sistemas, este directorio de usuarios es una organización lógica antes que física, ya que todos los archivos siguen estando físicamente en el mismo dispositivo.

Resumen sobre directorio de un nivel:

- Lista en la que todos los archivos están en el mismo directorio.
- Limitaciones: Si el número de archivos es muy elevado puesto que me obliga a usar nombres únicos.
- Ventajas: Puede gestionar los dispositivos del sistema como archivos.

### c). Directorio de dos Niveles (Two-Level Directory)

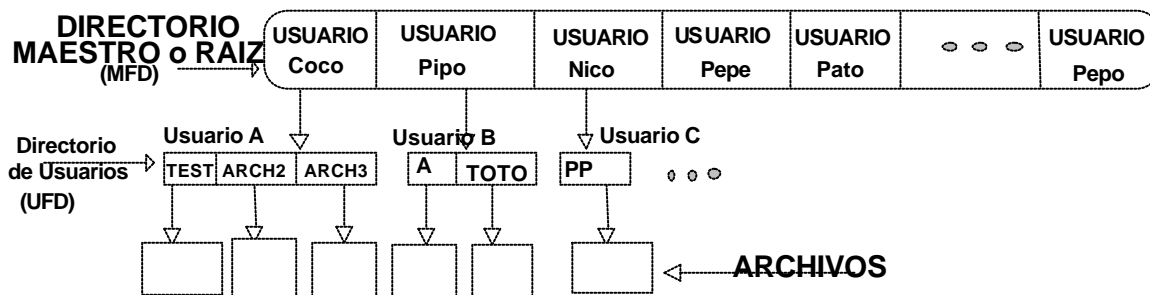


Figura 7.31 - Directorio de dos niveles

En una estructura de directorios de 2 niveles, cada usuario tiene su propio directorio de Archivos de Usuario (User File Directory UFD). Cada usuario tiene una estructura similar (lineal, binario o hashed), listando solo los archivos de él. Cuando un usuario inicia su trabajo o una sesión en la terminal, se lo busca en el Directorio de Archivos Maestro o Raíz (MFD) del sistema. El MFD se indexa por identificación de usuario y cada entrada apunta al UFD de ese usuario.

Los UFD deben ser creados y borrados. Para crearlo, un programa especial del sistema se ejecuta con el nombre de usuario e información contable. Este programa crea un nuevo UFD y una entrada en el MFD y para borrarlo se destruirán todos los punteros a los archivos (o no, depende la política de borrado que se emplee) para luego proceder a su eliminación.

**Ventajas:**

- Permite que varios usuarios trabajen con sus propios archivos sin interferir con los demás. De esta forma diferentes usuarios pueden tener archivos con el mismo nombre.
- Para crear o borrar archivos, el sistema operativo limita la búsqueda al UFD, de esta forma no se puede borrar accidentalmente un archivo de otro usuario.

**Desventajas:**

- Así como no deja que dos usuarios se interfieran, tampoco los deja trabajar en forma



conjunta. Si dos usuarios trabajan en el mismo proyecto, deberían poder compartir la información contenida en sus archivos.

- No permite agrupar archivos. Un usuario puede querer agrupar sus archivos, y no tenerlos todos juntos en una sola lista.

Veamos como resolver la desventaja de tener a los usuarios separados unos de otros. Si a un usuario se le permite el acceso a archivos de otros, debe existir algún mecanismo para poder hacer tal referencia.

Un directorio de dos niveles puede ser representado por un árbol de altura 2. La raíz es el Master File Directory, las ramas son los User File Directory y las hojas del árbol son los archivos en sí. La especificación de un nombre de usuario y un nombre de archivo define un **camino** en el árbol desde la raíz (MFD) hasta una hoja (el archivo especificado), así se define un **path name** (nombre de camino). Cada archivo en el sistema tiene un path name único que lo identifica. Por ejemplo si el usuario A quiere acceder al archivo TOTO del usuario B debería escribir: PEPE[USRB] o /USRB/TOTO. Cada sistema tiene su propia sintaxis para nominar archivos que están en otro directorio que no sea el propio del usuario.

Cuando el usuario hace referencia a un archivo en particular, solo se busca en su UFD. Así distintos usuarios pueden tener archivos con igual nombre, en tanto en su directorio ese nombre sea único. Para crear o borrar archivos de un usuario el sistema solo busca en la UFD de ese usuario. Incluso los directorios de usuarios pueden crearse o borrarse si es necesario. Existe un algoritmo especial que crea un nuevo UFD y agrega una entrada al MFD a pedido. Por supuesto la ejecución de este programa debe estar controlado por el S.O.. La asignación de espacio en disco para estos directorios puede manejarse con las técnicas ya discutidas en éste módulo.

Sin embargo, los problemas persisten con una estructura de este tipo. Este método efectivamente aísla cada usuario de otro. Esto es una ventaja cuando los usuarios son completamente independientes, pero también presenta como desventaja cuando los usuarios deben cooperar entre tareas y acceder archivos de otros usuarios. Algunos sistemas directamente no permiten el acceso de archivos ajenos. Otros, sí permiten el acceso a un archivo de otro directorio, para lo que debe darle la identificación del usuario y el nombre del archivo, para distinguir 2 archivos con el mismo nombre en distintos directorios.

Un caso especial sucede con los programas del sistema (utilitarios, compiladores, subrutinas, cargadores, etc.). Para no duplicar todos estos programas en cada directorio de usuario para su uso, se define un usuario especial, por ejemplo, USUARIO<sub>0</sub> donde se cargan todos los programas del sistema. Esto complica un poco las búsquedas pero soluciona el problema de espacio en disco. Al tratar un usuario ejecutar un programa, el SO busca primero en el directorio del usuario que invocó el programa y si no lo encuentra lo busca en ese directorio definido especialmente.

La secuencia de búsqueda de directorios cuando se nombra un archivo se conoce como **camino de búsqueda (search path)**

#### Resumen sobre Directorio de dos niveles:

- Usa estructuras de un nivel para cada usuario, lo que forma el directorio principal de archivos. (Master File Directory)
- El segundo nivel es la información de los archivos de cada usuario. (User File Directory)
- Existe un comando que permite crear un nuevo usuario, creando una nueva estructura en MFD y UFD.
- Los usuarios están aislados y no pueden compartir archivos. Excepcionalmente se permite acceder por medio del comando: Usuario \ archivo.
- Este sistema se puede usar para compartir la información de los propios archivos del sistema operativo. Esta información está almacenada en otra estructura del MFD como si fuera un usuario más, a la información que está en su correspondiente UFD pueden acceder todos los usuarios.
- Todo está controlado por el S.O.
- Existe la vía de acceso o PATH.

### d). Estructura de Árbol (Tree-Structured Directories)

Una vez comprendida la estructura de dos niveles como un árbol de dos niveles, la generalización natural es extender esta estructura a un árbol arbitrario de n niveles.

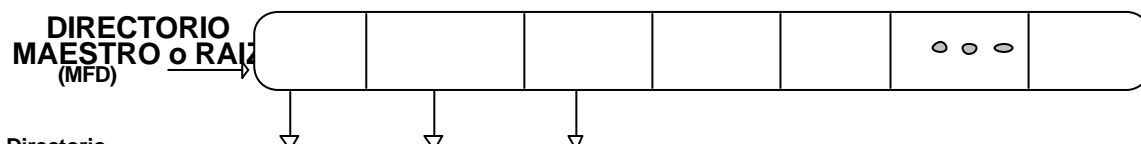
Este esquema permitirá a los usuarios crear sus propios subdirectorios y organizar los archivos como prefieran.

#### Ventajas:

- Todas las características mencionadas acerca de esta estructura constituyen ventajas sobre las estructuras anteriores.

#### Desventajas:

- No permite que los archivos o directorios sean compartidos por varios usuarios. Permite la duplicación de archivos que no es lo mismo. En el siguiente punto se tratará este tema.



Una vez que la estructura de directorios de 2 niveles es vista como un árbol de 2 niveles, la generalización es inmediata, extendiendo el concepto a un árbol de una cantidad de niveles arbitraria. Esto permite a los usuarios crear sus propios subdirectorios y organizar los archivos adecuadamente.

Un ejemplo de esta estructura lo da el S.O. UNIX. El árbol tiene un directorio raíz. Cada archivo en el sistema tiene un único nombre de camino, comenzando el mismo desde la raíz, atravesando todos los subdirectorios hasta llegar al nombre de archivo. Un directorio (o subdirectorio) contiene un conjunto de archivos y/o subdirectorios. Todos los directorios tienen el mismo formato interno. Un bit agregado a cada entrada de un directorio, define la misma como archivo (0) o subdirectorio(1). Se usan llamadas especiales al sistema para crear y borrar directorios.

Durante el uso normal, cada usuario tiene un directorio actual o efectivo de trabajo, el que contiene sus archivos y la mayoría de programas de su interés. Cuando se nombra un archivo, se realiza una búsqueda en ese directorio. Si no se encuentra, el usuario debe especificar o un nombre de camino o cambiar su directorio de trabajo efectivo.

Los nombres de camino pueden ser de dos tipos:

- Completos: Comienza en la raíz y sigue un camino hasta el archivo especificado.
- Relativos: Define un camino desde el directorio actual.

Un camino completo comienza en el directorio raíz y sigue con la lista de subdirectorios hasta llegar a un nombre de archivo. En cambio el camino relativo comienza a partir del directorio de trabajo. Por ejemplo, en el gráfico anterior si el directorio efectivo es usuario/test/mail, entonces el camino relativo arch1 referencia el mismo archivo dado por el camino completo usuario/test/mail

La habilidad de que el usuario defina su propia estructura de subdirectorios les permite a ellos imponer su propia estructura de archivos. Como resultado se puede tener directorios separados para archivos que están asociados por diferentes tópicos o distintos tipos de información (por ejemplo, el directorio programas puede contener todos los archivos fuente, etc.)

Un punto interesante es este método es como se va a manejar el borrado de un directorio. Si esta vacío, simplemente se puede borrar. Sin embargo, si no esta vacío, puede contener archivos e incluso subdirectorios. Para el borrado existen dos soluciones. Algunos sistemas no borrarán un directorio si no esta vacío. Esta solución puede conllevar una cantidad substancial de trabajo de borrando archivos. Una alternativa es suponer que ante un pedido de borrado de un directorio, debe borrarse archivos y subdirectorios contenidos.

Cualquiera de las dos soluciones es fácil de implementar y su elección es una cuestión de política. Existen sistemas que han implementado uno u otro método. El usuario B puede acceder archivos del usuario A, especificando el camino, ya sea completo o relativo. Alternativamente, el usuario B puede cambiar su directorio efectivo de trabajo por el del usuario A y acceder directamente por nombre de archivo.

Algunos sistemas permiten a usuarios definir sus propios caminos de búsqueda. En este caso, el usuario B puede definir un camino de búsqueda que sea (1) su directorio local, (2) el directorio de archivos del sistema y (3) el directorio del usuario A, en este orden. En tanto el nombre del archivo del usuario A no sea el mismo que el de un archivo de B o del sistema, B puede llamar directamente por el nombre de archivo para acceder a A.

#### **Resumen sobre directorios en árbol:**

- El punto de partida es el directorio raíz.
- Todos los directorios tienen la misma estructura y cada uno de ellos puede contener archivos o directorios, para que el sistema los distinga les asigna un bit (1 Directorio, 0 Archivo)
- Directorio Actual: Punto único en el que estoy trabajando.
- Vía de acceso: Relativa o Absoluta (desde la base del árbol)

### e). Estructura de Gráfos Acíclicos (Acyclic Graph Directories)

Supongamos que hay dos programadores trabajando en el mismo proyecto. Los archivos asociados a ese proyecto pueden ser almacenados en un subdirectorio separado de otros proyectos y archivos de los dos programadores. Pero como ambos son responsables de este proyecto, los dos quieren que este directorio les pertenezca. Es decir, el directorio común debería ser **compartido**. Un directorio o archivo compartido estará repetido dos o más veces en el File System.

Nótese que un archivo o directorio compartido no es lo mismo que tener dos copias del mismo. Con dos copias, cada programador puede ver la copia en vez del original, y si un programador cambia el archivo estos cambios no aparecerán en la otra copia. Con archivos compartidos existe un solo ejemplar del archivo y así cualquier cambio hecha por una persona será inmediatamente visible a la otra. Esto es particularmente útil para directorios compartidos; un archivo creado por una persona aparecerá automáticamente en todos los subdirectorios compartidos.

Remarquemos, que un directorio o archivo compartido, no es lo mismo que tener dos copias. Con el uso de copias los cambios introducidos por uno no aparecen en la copia del otro. Si se comparte, si aparecerán los cambios.

Con una estructura de árbol no se pueden compartir de esta forma los archivos o directorios. En cambio un *gráfico acíclico* permite que los directorios tengan subdirectorios y archivos compartidos.

Una estructura de árbol inhibe el uso de directorios o archivos compartidos, lo que si facilita una estructura de directorio de **grafos acíclicos (acyclic graph)**. Este método basado en un *grafo acíclico* (es decir, un grafo sin ciclos) es una generalización natural de las estructuras de árbol.

La creación de archivos y subdirectorios compartidos puede implementarse de varias maneras. Una forma muy común es crear una nueva entrada a un directorio que se la conoce como **link**. Un link es un puntero a otro archivo a subdirectorio.

#### Vínculos Simbólicos (Symbolic Link)

Se crea una entrada nueva en el directorio llamada vínculo (link). Un vínculo es un puntero a otro archivo o subdirectorio. Por ejemplo, un vínculo podría ser implementado por un *nombre de camino completo* (un vínculo simbólico). Ejemplo el subdirectorio A. El vínculo es resuelto usando el path name para ubicar el archivo real.

#### Entradas de Directorio Duplicadas (Duplicated Entries)

Se duplica toda la entrada de directorio del archivo a duplicar en ambos subdirectorios. Así ambas entradas son idénticas, lo cuál hace que el original y la copia sea indistinguibles. Una severa restricción con la duplicación de entradas es mantener la consistencia cuando se modifica el archivo.

##### Ventajas:

- La estructura de gráficos acíclicos es mucho más flexible que la estructura de árbol.
- Permite compartir archivos y/o subdirectorios.

##### Desventajas:

- El sistema operativo debe asegurarse que no existan ciclos en los vínculos.
- Es compleja la implementación.
- Distintos nombres de archivo pueden referirse al mismo archivo, con lo cuál cualquier actividad contable, backup, búsqueda de archivos, etc. se complica ya que hay que vigilar de no contabilizar dos veces el mismo archivo.
- Cuando se borra un archivo compartido, la actualización de los vínculos es compleja.

En un sistema donde se comparte mediante vínculos simbólicos, el borrado de un vínculo no afecta el archivo original; sólo se borra el vínculo. Ahora, si se borra el archivo quedan los vínculos sueltos. Se los puede buscar y removerlos, pero a no ser que se mantenga una lista de vínculos asociados con el archivo, esta búsqueda puede ser muy costosa.

Alternativamente podemos dejar los vínculos sueltos hasta que un usuario intente usarlo. En ese momento el sistema operativo determina que el archivo no existe y devuelve un mensaje de error.

Se presentan otras alternativas como preservar el archivo hasta que todas las referencias al mismo sean eliminadas. Pero este mecanismo es complejo y suele ocupar mucho espacio de almacenamiento.

Un link puede implementarse como un camino completo (link simbólico). Acá, cuando referencia un archivo, se busca en el directorio. La entrada en ese directorio esta marcada como un link con el camino de la ubicación efectiva del archivo. Los links son fácilmente identificados en la entrada al directorio y son punteros nominados indirectos.

Una estructura de este tipo es más flexible que una simple de árbol, pero también es más compleja. Existen varios problemas que deben ser cuidadosamente tratados. Notemos que ahora puede existir un mismo archivo con más de un nombre de camino. En otras palabras, distintos nombres

de archivos pueden estar refiriéndose al mismo.

Otro problema surge con el borrado. Una posibilidad es borrar cuando alguien lo pide, pero esta acción, como dijimos, puede dejar a punteros apuntando a un archivo ya inexistente. mas aun, si los punteros apuntan a direcciones en disco y el espacio fue re-usado, estos punteros pueden ahora estar apuntando al medio de algún otro archivo.

En un sistema donde compartir archivos se resuelve con links, este problema es fácil de manejar. El borrado de un link no implica necesariamente borrar el archivo original, solo se remueve el link. Si se borra el archivo, se libera espacio y el puntero queda "colgado". Se pueden buscar luego estos links para borrarlos, pero esto puede tomar bastante tiempo. Una solución es dejarlos hasta que se realice un intento de uso. En ese momento, el S.O. lo resuelve como si fuera el caso de acceder a un archivo inexistente.

Otra solución es no borrar el archivo hasta que todos los links sean previamente borrados.

## Resumen sobre Grafos Acíclicos:

- Expresamente diseñado para compartir información, en este tipo un mismo archivo o un mismo directorio físico, puede aparecer en los directorios de dos o más usuarios.
- Si alguien modifica un archivo que está compartido también se modifica para el otro usuario.
  - UNIX:
- Existe un enlace simbólico de un enlace en otro.
- Es una estructura muy flexible y compleja.
- Sólo se borran los archivos si se han borrado todas sus referencias.
- Primero se hace una exploración exhaustiva (dos caminos llevan a un mismo archivo dos veces) y luego una eliminación de archivos (no pueden quedar referencias sueltas)

### f) Directorio de Gráfos Generalizado.

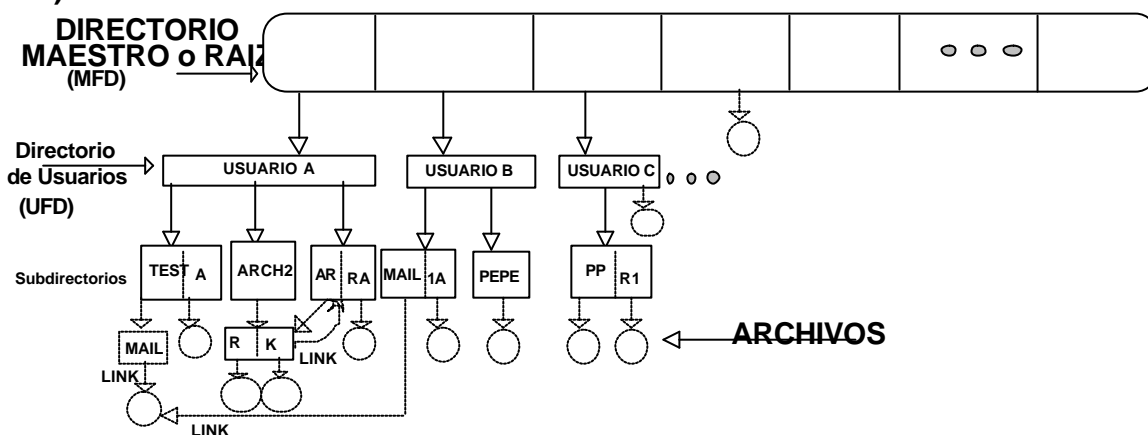


Figura 7.33 - Directorio con estructura Gráfo General

Un serio problema del uso de una estructura de grafos acíclico es asegurarse de la no existencia de ciclos. Si se arranca con una estructura de directorios de 2 niveles y permitimos a los usuarios crear subdirectorios, el resultado es una estructura de árbol. Sin embargo al agregar links, se destruye el árbol, resultando una estructura simple de grafo.

Se desea evitar, ante una referencia a un archivo, la búsqueda de dos veces por un subdirectorio compartido, por razones de performance. Si se permiten ciclos, con un algoritmo pobre de búsqueda, podemos desembocar en una búsqueda en círculo infinito y nunca terminar.

Por ende, una estructura de grafo acíclico es mucho más fácil de trabajar.

## Resumen sobre Grafo General:

- No hay una organización explícita, pueden aparecer ciclos.
- Necesitan una construcción especial, necesitan un elemento que atravesase todo el sistema de archivos para evitar el problema de las autoreferencias (cuando un el número de referencias es distinto de cero y físicamente no es posible acceder al archivo)
- Para solucionar esto se examina todo el sistema de archivos, aquel archivo al que se pueda acceder queda marcado y luego se borran los que no estén marcados. Esto se conoce como **Garbage collection**. Es un sistema lento y pesado.

### 7.7.2.- Nominación (naming)

Lo que los usuarios necesitan es referenciar un archivo por su nombre. Claramente dentro del sistema cada archivo debe poseer un nombre no ambiguo, en un sistema compartido es inaceptable poner en cada usuario la carga de asignar nombres únicos en el sistema. El uso de una estructura arbórea de directorios minimiza la dificultad de asignar nombres únicos. Cualquier archivo residente en el sistema se puede localizar siguiendo un camino desde la raíz (directorio maestro) bajando varias ramas hasta alcanzar el archivo. La serie de nombres de directorios (ramas) constituye un **Pathname** (nombre del camino) del archivo. El nombre del directorio maestro esta implícito, pues todos los caminos comienzan en ese directorio. Ahora se acepta que el sistema haya muchos archivos con el mismo nombre **mientras** tengan pathname únicos.

Con la estructura arbórea nace el concepto de directorio de trabajo (**working directory**) que es la rama en la que actualmente se está posicionado. Los archivos se referencian relativamente al working directory (si no se especifica explícitamente el pathname). Cuando se crea un proceso o cuando un usuario se loguea al sistema el directorio default para su working directory es el directorio de usuario. Durante una sesión de trabajo, el usuario puede navegar (hacia arriba o hacia abajo del árbol) para definir distintos directorios de trabajo.

## 7.8. PROTECCIÓN DE ARCHIVOS.

Cuando se almacena información en un soporte de un computadora, uno de los mayores cuidados a tenerse en cuenta por su importancia es la protección contra daños físicos (confiabilidad) y accesos inapropiados o no permitidos (protección).

La confiabilidad se logra generalmente haciendo copias duplicadas de los archivos. Muchos sistemas operativos copian automáticamente los archivos de disco a cinta cada intervalo regular de tiempo (una vez al día, a la semana o al mes. Dependiendo de cómo se los haya programado).

El sistema de archivos puede destruirse por fallas de hardware, tales como error en lectura o grabación, fallas de alimentación de energía eléctrica, aterrizaje de cabezas, polvo, humedad, temperatura, vandalismo, etc.. Problemas con el software del sistema de archivos también puede desembocar en pérdida de información.

La protección puede proveerse de varias formas. Para pequeños sistemas, por ejemplo, guardando los disquetes en lugar seguro. Para sistemas grandes, se usan otros mecanismos de protección.

La necesidad de proteger archivos surge de la habilidad del acceso a los mismos. En sistemas que no permiten el acceso de otros usuarios a archivos propios esta protección no es necesaria.

Un límite extremo de protección sería prohibir totalmente el acceso de otros usuarios y el otro extremo proveer libre acceso sin restricciones. Ambos métodos son demasiado extremos para un uso general. Lo que se necesita es un acceso controlado.

Los mecanismos de protección proveen acceso controlado limitando los tipos de acceso que pueden realizarse a archivos. El acceso es permitido o prohibido de acuerdo a varios factores, por lo que se otorgan "permisos" a los usuarios para dichos accesos.

Las operaciones que pueden controlarse son:

- Lectura: leer de un archivo.
- Escritura: Escribir o re-escribir un archivo.
- Ejecución: Cargar el archivo a memoria y ejecutarlo.
- Agregar: Escribir nueva información al final del archivo.
- Borrar: Borrar un archivo y liberar el espacio para su re-uso.

Otras operaciones, tales como renombrar o copiar o editar un archivo también pueden ser controladas. Aunque generalmente, por ejemplo el copiado de un archivo requiere múltiples lecturas sobre el mismo, en este caso un usuario con permisos de acceso libre de lectura, también puede copiar el archivo, imprimirlo, etc.

Las operaciones sobre directorios que deben ser protegidas son algo diferentes. Se desea controlar la creación y borrado de archivos en un directorio. Adicionalmente se puede necesitar controlar si un usuario va a determinar o no la existencia de un archivo en un directorio. A veces, el conocimiento de la existencia y nombre de un archivo puede resultar importante. Así el listado del contenido de un directorio debe ser también una operación protegida. Además se puede controlar si un usuario puede o no saber de la existencia de un archivo en el directorio (archivos ocultos).

Muchos mecanismos diferentes de protección han sido propuestos. Como siempre, cada uno tiene sus ventajas y desventajas y deben seleccionarse en forma apropiada teniendo en cuenta la aplicación pretendida.

Un sistema pequeño con pocos usuarios no necesitará seguramente los mismos tipos de protección que un gran computador usado en una gran corporación que es empleado para

investigación, finanzas y personal, por ejemplo.

La protección puede proveerse de muchas formas. La necesidad de proteger archivos es un resultado directo de la facilidad de acceso a los mismos. En sistemas que no permite el acceso a archivos de otros usuarios, la protección no es necesaria ya que la misma estructura del sistema prohíbe el acceso a otras personas.

Los mecanismos de protección proveen acceso controlado limitando los tipos de acceso que se pueden hacer. El acceso es permitido o negado dependiendo de los permisos concedidos, para distintos tipos de operaciones que pueden ser controlados: lectura, escritura, ejecución, borrado, renombrar el archivo, copiarlo, editarlo, etc.

Para muchos sistemas, las funciones de alto nivel como copiado pueden ser implementadas por programas del sistema que a su vez hacen System Calls de bajo nivel. La protección sólo se implementa a bajo nivel. Por ejemplo, el copiado de un archivo se puede implementar como una secuencia de lecturas. En este caso un usuario con permiso de lectura también puede copiar el archivo, etc.

### 7.8.1. Nombre

Los esquemas de protección de varios sistemas dependen de la imposibilidad del usuario de acceder a un archivo que no sepa su nombre. Si un usuario no puede nominar el archivo, entonces no puede operar sobre el mismo. Este esquema asume que no existe un mecanismo para la obtención de los nombres de los archivos de los otros usuarios y que no puedan ser fácilmente adivinados. Ya que los nombres son generalmente elegidos como representación de la información que contienen, a menudo pueden ser fácilmente adivinados. Un mecanismo muy simple de protección es ocultar el nombre.

### 7.8.2. Contraseñas (Passwords)

Otra solución es asociar una contraseña a cada archivo. Igual que el acceso a un sistema computador es a menudo controlado por contraseñas, el acceso a cada archivo también puede ser controlado por contraseñas.

A cada archivo se le asocia un password. Si los passwords se eligen aleatoriamente y se modifican a menudo, este esquema puede resultar conveniente. Sin embargo esta protección es del tipo "todos o ninguno", es decir hay una sola palabra clave que da acceso completo a un archivo. Para proteger a un nivel más detallado se necesitan múltiples passwords.

Comúnmente, sin embargo, solo una contraseña es asociada a cada archivo. Esta protección así se convierte en una del tipo pasa-no pasa. Para dar protección a un nivel más detallado, se necesitan de múltiples contraseñas.

### 7.8.3. Control de Acceso

El acceso se hace dependiendo la identidad del usuario. Una lista de acceso (access list) puede estar asociada con cada archivo y directorio especificando el nombre del usuario y los tipos de accesos permitidos (permisos). Cuando un usuario pide un determinado archivo, el sistema operativo verifica su lista de acceso. Si el usuario está en la lista de acceso, le permite el uso, caso contrario se genera una violación de protección y el proceso es finalizado o se toman otras medidas.

Una versión condensada de este esquema se usa en muchos sistemas. El mayor problema con las listas de acceso puede ser la longitud que pueden alcanzar las mismas. Si deseamos que todos los usuarios puedan leer un archivo, deben listarse a todos con el acceso de lectura. Para condensar la longitud de la lista de accesos, muchos sistemas reconocen 3 clasificaciones de usuarios: propietario, grupo y cualquier otro usuario. Todo archivo tiene un propietario: su creador.

Adicionalmente, el propietario puede pertenecer a un grupo de usuarios que trabajan juntos y necesitan compartir el archivo y necesitan similar tipo de acceso. Por ejemplo, los miembros de un grupo de programadores, pueden definir un grupo. Finalmente, están los demás usuarios.

Con esta clasificación, se necesitan solo 3 campos para definir las protecciones. Cada campo es un conjunto de bits que permiten o no el acceso asociado a cada bit. Por ejemplo, el S.O. UNIX define 3 campos de 3 bits cada uno: rwx, donde r controla el acceso de lectura, w controla el acceso de escritura y x controla la posibilidad de ejecución. Un campo separado se guarda para el propietario del archivo, para su grupo y para los demás usuarios. Con este esquema se necesitan 9 bits por archivo para almacenar la información de protección.

La protección puede estar asociada ya sea para el archivo como para el camino usado para nominar al archivo. El método más común provee protección al camino. Así si un camino nombra un archivo en un directorio, el usuario debe tener acceso al directorio y al archivo.

#### 7.7.4.- El compartir Archivos ( FILE SHARING)

En un sistema multiusuario, casi siempre existe la necesidad de permitir a los usuarios compartir archivos. Dos problemas surgen:

1. Los derechos de accesos
2. Gestión de los accesos simultáneos

##### Derechos de Acceso:

El sistema de archivos provee una herramienta flexible para permitir compartir extensos archivos entre los usuarios. El sistema de archivos debe proporcionar un numero de opciones de modo en que un archivo que es accedido pueda ser controlado. Normalmente, al usuarios o a los grupos de usuarios se les otorgan ciertos derechos de acceso a cada archivo. Un amplio rango de derechos de acceso se ha venido usando. La siguiente lista representa los derechos de acceso que pueden ser asignados a un usuario en particular para un archivo en particular:

1. Ninguno (none): El usuario no puede siquiera determinar la existencia del archivo ni mucho menos acceder al mismo. No se permite al usuario leer el directorio de usuario que incluya al archivo.
2. Conocimiento (knowledge): El usuario sabe de la existencia del archivo y quien es el dueño. El usuario puede solicitar los derechos de acceso adicionales al propietario.
3. Ejecución (Execution): El usuario puede ejecutar y cargar un programa pero no copiarlo (Los programas propietarios usan esta restricción).
4. Lectura (Reading): El usuario puede leer el archivo para cualquier propósito, incluyendo copia y ejecución.
5. Adición (Append): El usuario puede añadir datos al archivo, generalmente al final, pero no puede modificar o borrar el contenido del mismo.
6. Actualización: El usuario puede modificar, borrar y añadir otros datos al archivo. Este permiso incluye permiso para poder escribir el archivo desde el inicio (archivo vacío), reescribirlo completamente o en parte y borrar todo o parte del archivo. Algunos sistemas distinguen entre distinto grado de actualización.
7. Cambio de protección: El usuario puede cambiar los derechos de acceso otorgados a usuarios.
8. Borrado: El usuario puede borrar el archivo del sistema de archivos.

Se consideran que estos permisos son jerárquicos, si posee un permiso de grado  $n$  implícitamente posee los permisos de  $n-1$ . El dueño (Owner) es la persona (usuario) que crea el archivo. El dueño posee todos los derechos previamente enunciados y puede otorgar permisos a los demás.

El propietario de un archivo dispone de los derechos de acceso listados antes y puede otorgar derechos a los otros. Puede ofrecerse acceso a las siguientes clases de usuarios:

1. Usuario específico: Usuarios individuales quienes son designados por su ID de usuario.
2. Grupos de usuarios: Un conjunto de usuarios no definidos individualmente.
3. Todos: Todos los usuarios que tengan acceso al sistema. Estos serán archivos públicos.

##### Acceso Simultáneos:

Cuando el acceso es concedido para añadir o actualizar un archivo a mas de un usuario, el sistema operativo o el sistema de gestión de archivos debe hacer cumplir una disciplina. Un método de fuerza bruta consiste en permitir a los usuarios bloquear el archivo entero cuando lo vaya a actualizar. Un mejor control es bloquear los registros individuales durante la actualización. Al disertar la posibilidad de accesos comparados, deben abordarse aspectos de exclusión mutua e ínter bloqueo.

#### 7.8.4.-CONFIABILIDAD

Veamos un algoritmo para evitar que ocurra lo siguiente:

1. El usuario A pide una asignación extra para un archivo existente.
2. Se satisface el pedido y las tablas de asignación de archivo y de disco (o sea la FAT del archivo y la DAT (Disk Allocation Table) del disco) se actualizan inmediatamente en memoria, pero no en disco.
3. El sistema "se cae" (crash) y luego se lo reinicia.
4. El usuario B pide una asignación extra de espacio para un archivo suyo ya existente. El pedido se satisface asignándole espacio en disco que se superpone (overlap) con la asignación que se le hizo al usuario A. Se le dio al archivo B el mismo bloque que se le dio antes a A. Ambas FAT poseen un mismo numero de bloque.

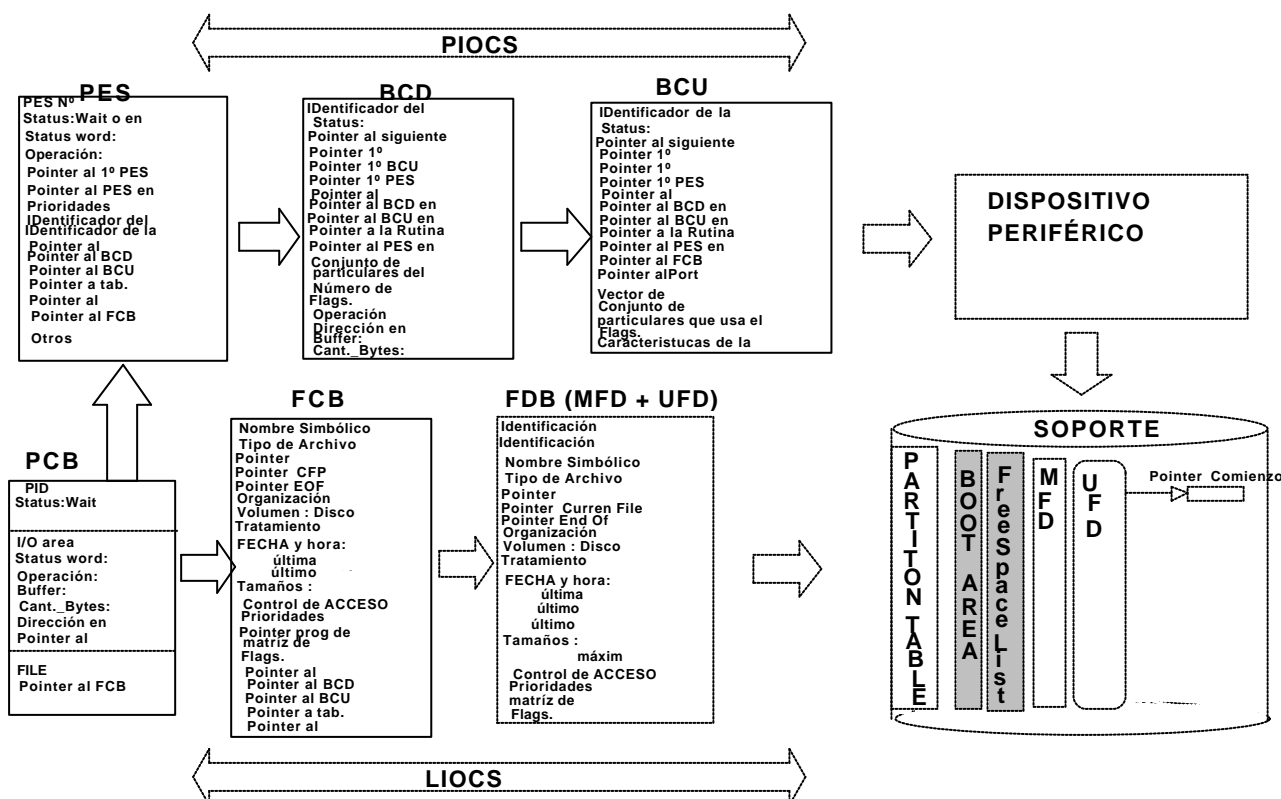
El problema lo ocasiona el hecho que por cuestiones de eficiencia, el sistema mantiene en memoria central una copia de la DAT y otra de la FAT. Para evitar este problema se deben llevar adelante los siguientes pasos cada vez que se requiera una asignación de espacio a un archivo:

1. Cierre (Lock) la DAT en disco. Esto evita que otro usuario pueda alterar la tabla (hasta que la asignación se complete ).
2. Buscar en la DAT el espacio disponible (mayor o igual al requerido). Esto presupone que se tiene en memoria central una copia de la DAT. Si esto no es así, primero debo cargar la DAT en memoria central y luego “buscar” el espacio solicitado.
3. Asignar el espacio solicitado, actualizar la DAT en memoria central y luego actualizar el disco (regrabar la DAT en el disco). Para el caso de asignación de espacio en disco encadenada, esto involucra (incluye) actualizar algunos punteros en el disco.
4. Actualizar la FAT del archivo y actualizar el disco (regrabar la FAT correspondiente).
5. Desbloquear (UnLock) la DAT.

Si bien esta técnica evita errores, el impacto en el rendimiento (performance) es importante cuando se asigna porciones pequeñas. Para reducir la sobrecarga (overhead) se podría usar un esquema de asignación con almacenamiento diferido (batch storage). En este caso, se obtiene un lote de porciones libres del disco para ser asignado. Dichas porciones se marcan en uso (not free). Las asignaciones usando este lote, se pueden efectuar en memoria central. Cuando el nuevo lote se agota, la DAT se actualiza en el disco y se adquiere un nuevo lote. De ocurrir una caída del sistema, las porciones de disco marcadas “en uso” deberían limpiarse (clean up) de alguna manera antes de poder ser reasignadas. La técnica del “limpiado” dependerá de las características particulares del sistema de archivos (file system).

## 7.9. Métodos de implementación del Sistema de Archivos.

Un sistema de archivos tiene dos problemas de diseño muy diferenciados desde el aspecto lógico. El primer problema es la definición de como el sistema de archivos será visto por el usuario. Esto implica la definición de archivos y sus atributos, operaciones permitidas sobre el mismo y la estructura de directorios. El segundo problema tiene que ver con la implementación de la estructura de datos y de algoritmos para el mapeo del sistema de archivos lógico a los dispositivos físicos. Otro gran problema radica en el aspecto físico de la vinculación para el acceso. Esto se resuelve mediante tablas como se ve en la **figura 7.34**.



**Fig. 7.34** Distintas tablas que intervienen en la vinculación de un archivo.

En la **Figura 7.35** presentamos en esquema jerárquico de servicios involucrado en el uso de un



archivo. En este esquema cada nivel usa los servicios de los niveles inferiores, para a su vez realizar funciones que servirán a niveles superiores. El menor nivel, Control de E/S, consiste en los drivers de dispositivos y manejadores de interrupciones, que son los encargados de transferir información entre la memoria central y el sistema de discos. El Sistema de Archivos Básico usa el nivel anterior para leer y escribir bloques en particular del disco. Cada bloque es identificado por su dirección numérica (por ejemplo, unidad 1, cilindro 73, superficie 2, sector 10).

El Módulo de organización de archivos maneja tanto bloques de disco como archivos. Conociendo el tipo de asignación usado y la dirección del archivo en disco, puede generar direcciones de los bloques a ser leídos por el Sistema de Archivos básico. Finalmente, el Sistema de Archivos Lógico usa la estructura de directorios para brindar al Módulo de Organización de Archivos con los parámetros necesarios de un archivo con nombre simbólico.

Para crear un archivo, el programa de aplicación llama al Sistema de Archivos Lógico. Este conoce el formato de la estructura de directorios. Se lee el directorio apropiado en memoria, se actualiza con la nueva entrada y se lo escribe en disco. Los directorios pueden tratarse igual que archivos, con un campo agregado para indicar que son directorios. Así el Sistema de Archivos Lógico puede llamar al Módulo de Organización de Archivos que mapea la E/S al directorio dando un número de bloque en disco, el que es pasado al Sistema Básico de Archivos y al Sistema de Control de E/S.

Una vez que el directorio ha sido actualizado, el Sistema de Archivos Lógico puede usarlo para realizar E/S. Cuando se abre un archivo, se realiza una búsqueda en la estructura de directorios para ubicar la entrada correspondiente.

Es posible realizar esta búsqueda por cada operación de E/S, pero resulta demasiado ineficiente. Para acelerar la búsqueda, el S.O. guarda en memoria central una tabla de archivos abiertos. Solo se realiza la búsqueda en la tabla de directorios con la primera referencia al archivo. Cuando se cierra el archivo se actualiza la entrada a la tabla de directorio en disco.

Este método, aunque más veloz, puede causar problemas. Si falla el S.O., normalmente se pierde la tabla de archivos abiertos en memoria y cualquier cambio en los directorios de archivos abiertos. Esto puede dejar el sistema de archivos en un estado inconsistente, donde la estructura efectiva de los archivos en disco no concuerdan con la descripción de los mismos en la entrada del directorio.

Otro aspecto importante en el uso de archivos es el control de accesos (protección). Cada acceso a un archivo o directorio debe ser controlado como se explicó en el punto anterior.

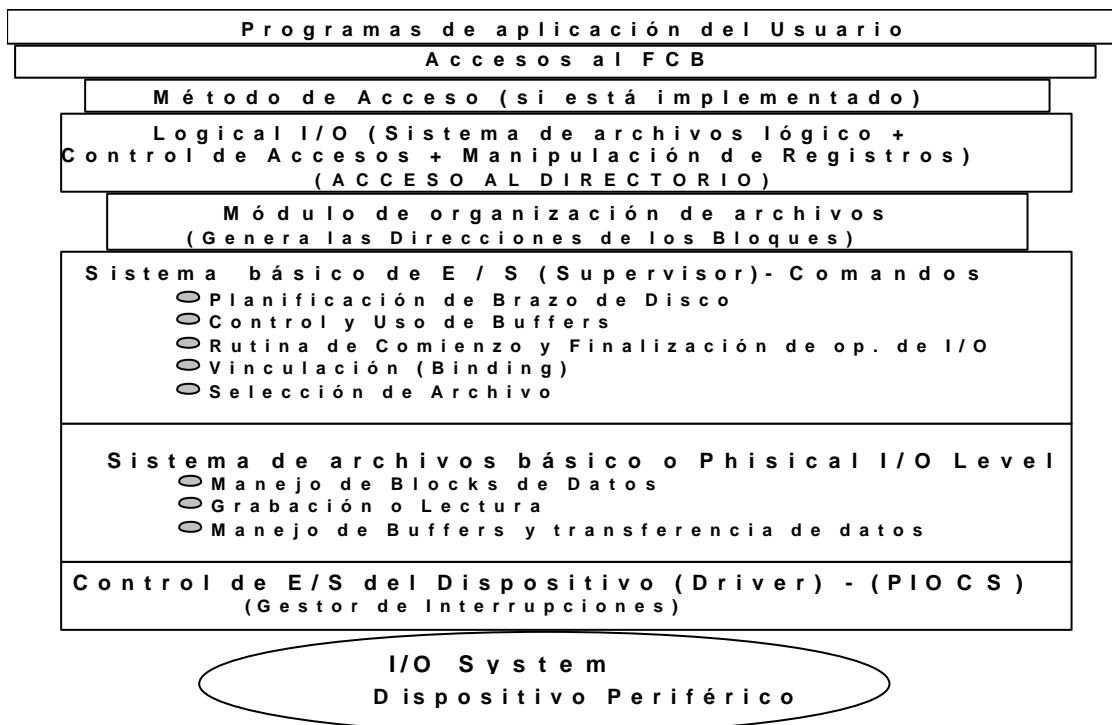


Fig. 7.35 Distintos niveles en un Sistema de Gestión de Archivos.

### 7.9.1.-Arquitectura del Sistema de Archivos

Veremos un esbozo de una organización de software TÍPICA, recordando que diferentes Sistemas operativos pueden utilizar diferentes organizaciones.

**a) Al nivel mas bajo los conductores de dispositivos (device drivers)**

Se comunican directamente con los dispositivos, o sus controladores o su canal.

El Device driver es responsable de

- 1- Comenzar las operaciones de E/S en un dispositivo.
- 2- Procesar en forma completa un pedido de E/S.

Para operaciones de archivos los principales Device drivers involucrados son los de cintas y los de discos magnéticos. A los Device drivers se los considera parte del Sistema operativo.

**b) Un nivel mas arriba se encuentra el Sistema de Archivos básico (o E/S Física, Plocs)**

Es la interfase primaria con el dispositivo (environment), fuera del computador. Se maneja con el intercambio de bloques de datos entre la memoria principal y el soporte de almacenamiento.

Se ocupa tanto de la colocación de esos bloques sobre el dispositivo de almacenamiento secundario como del buffering de esos bloques en memoria principal.

No sabe para nada acerca del contenido de los datos ni de la estructura de los archivos involucrados.

El sistema de archivos básico a veces se considera parte del Sistema operativo.

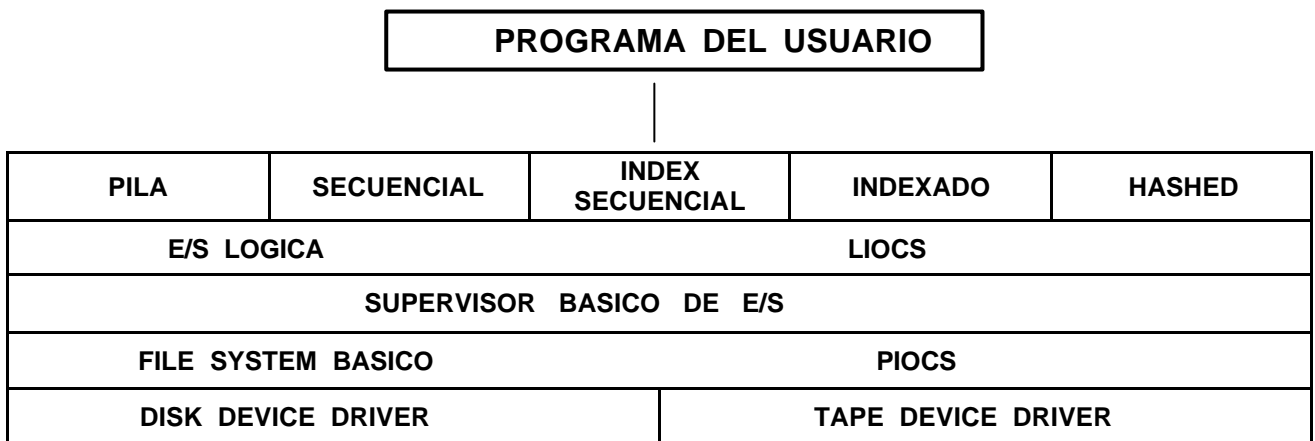


Figura 7.36 Arquitectura de un File system.

**c) Supervisor básico de E/S**

Es el responsable de la iniciación y la terminación de toda E/S de archivos. En este nivel se mantienen las estructuras de control relacionadas con la E/S del dispositivo, la planificación y el estado del archivo (file status). Este módulo es quien selecciona sobre qué dispositivo se efectuará la E/S en base a qué archivo ha seleccionado este módulo. También se ocupa de la planificación de los discos y de las cintas a fin de optimizar la performance. En este nivel se asignan los buffers de E/S y también se asigna (allocate) memoria secundaria (o sea espacio en disco). El supervisor básico de E/S es aparte del sistema operativo.

**d) La E/S Lógica (o Liocs)**

Permite que usuarios y aplicaciones accedan a los **REGISTROS** del archivo. Mientras que el **PIOCS** opera con bloques (ó sectores) de datos, el **LIOCS** opera con los registros (records) del archivo.

Provee de una capacidad de propósito general para la **E/S de REGISTROS** y mantiene los datos básicos de cada archivo.

PIOCS= PHYSICAL INPUT OUTPUT CONTROL SYSTEM

LIOCS= LOGICAL I/O CONTROL SYSTEM

**e) El nivel del File System mas cercano al usuario**

Se denomina generalmente Método de Acceso. Provee de una interfase standard entre las aplicaciones y el file system (y los dispositivos) que contiene los datos. Los diferentes métodos de acceso reflejan las diferentes estructuras (organizaciones) y las diferentes formas de acceder y procesar los datos.

Como resumen de la implementación de un File System se tiene un orden ascendente a lo más próximo al usuario y en descendente desde lo más básico y físico:

**1. Drivers:**

- Controla cada uno de los dispositivos que se va a usar.
- Inicia y termina las operaciones de E/S.
- Código en modo superusuario.

**2. FS/Básico:**

- Nivel físico de E/S
- Se gestionan bloques de datos (unidad de información en almacenamiento)
- Se gestionan las zonas de almacenamiento interno (Buffers)

**3. Supervisor de E/S:**

- Inicia y termina las operaciones con archivos
- Se gestionan las estructuras de control que permiten efectuar procesos de E/S.
- Planificación de dispositivos (ahorrar tiempo)
- Reserva de espacio en Memoria Secundaria.

**4. E/S Lógica:**

- Permite a los usuarios y aplicaciones acceder a los registros
- No trabaja con archivos, sino con registros.

**Agrupación de registros (record blocking)**

Para realizar E/S, los registros deben organizarse en bloques. Dado un tamaño de bloque, pueden seguirse los siguientes tres métodos de agrupación en bloques:

1. Bloques fijos: Se usan registros de longitud fija y un número entero de registros son Guardados en un bloque. Puede haber espacio sin usar al final de cada bloque.
2. Bloque de longitud variable por tramos: Se usan registros de longitud variable y agrupada en bloques sin dejar espacios sin usar.
3. Bloque de longitud variable sin tramos: Son usados registros de longitud variable, pero no se dividen en tramos. En la mayoría de los bloques habrá un espacio desperdiciado, debido a la imposibilidad de aprovechar el resto del bloque si el registro siguiente es mayor que el espacio sin usar restante.

Los bloques de tamaño fijo son el modo más común de archivos secuenciales con registro de longitud variable. Los bloques de longitud variable por tramos constituyen un almacenamiento eficaz y no ponen límites al tamaño de los registros. Pero esta técnica es difícil de implementar.

**Fiabilidad**

Considérese el siguiente escenario:

1. El usuario A solicita una asignación para añadir datos a un archivo existente.
2. La petición se atiende y se actualizan en memoria central las tablas de asignación de disco y archivos, pero no aun en el disco.
3. El sistema se cae y a continuación se reinicia
4. El usuario B solicita una asignación y se le otorga un espacio en el disco que se solapa con la última asignación hecha al usuario A.
5. El usuario A accede a la sección solapada mediante una referencia que esta almacenada en el archivo de A.

Esto surge debido al que el sistema mantiene copias de la tabla de asignación de disco y la tabla de asignación de archivos en memoria central. Para evitar esto puede seguir los siguientes pasos:

1. bloquear en el disco la tabla de asignación de disco
- 2- Buscar espacio disponible en la tabla de asignación de disco.
- 3- Asignar el espacio, actualizar la tabla de asignación de disco y actualizar el disco.
4. Actualizar la tabla de asignación de archivos y actualizar el disco.
5. Desbloquear la tabla de asignación de disco.

## **7.10. Bibliografía recomendada para este módulo**

1. Operating Systems. (Second Edition), Stallings Willams; Prentice Hall, Englewood Cliff, NJ. 1995, 702 pages. [Sta 95 ]
2. Operating Systems Concepts (Fifth Edition), Silberschatz, A. and Galvin P. B; Addison Wesley 1994 850 Pages. [Sil 98]
3. Operating Systems Concepts and Design (Second Edition), Milenkovic, Millan; Mc Graw Hill 1992
- Modern Operating Systems, Tanenbaum, Andrew S.; Prentice - Hall International 1992, 720 pag. [Tan92]
5. Operating Systems, Design and Implementation, Tanenbaum, Andrew S.; Prentice - Hall International, 1987, 800 pag.
6. Fundamentals of Operating Systems, Lister, A.M. ; Macmillan, London 1979
7. Operating System Design - The XINU Approach, Comer, Douglas; Prentice Hall 1984

8. Operating System, Lorin, H., Deitel, H.M.; Addison Wesley; 1981;
9. A discipline of Programming, Dijkstra, Edward W.; Englewood Cliffs, Prentice Hall; 1976
10. The Art of computer Programming (Volumen 1, 2, y 3); Knuth, Donald; Addison - Wesley Publishing Co.; 1974
11. Operating Systems: Structures and Mechanisms; Jensen Philippe; Academic Press 1985
12. The UNIX Operating System, Christian, K.; John Wiley; 1983
13. UNIX System Architecture, Andleigh, Prabhat K.; Prentice - Hall International; 1990
14. The UNIX Programming Environment, Kernighan, B.W. and Pike, R.; Prentice - Hall International 1984
15. The UNIX System V Environment, Bourne, Stephen R.; Addison Wesley; 1987
16. UNIX Utilities a Programmer's Guide.; Tare, R. S.; Mc Graw-Hill.; 640 pag.
17. Tricks of the UNIX Masters. ; Sage, R. S.;
18. UNIX System Readings and Applications. (Vol I y II); AT&T Bell Laboratories.; Prentice Hall Englewood Cliffs; 1987- 416 pag.
19. Introducing UNIX System V; Morgan, R. and McGilton, H.; Mc Graw Hill; 480 pag.
20. Sistemas Operativos Conceptos y diseños.(Segunda Edición); Milenkovic Milan; Mc Graw Hill; 1994. [Me194]
21. Sistemas de Explotación de Computadores; CROCUS; Paraninfo; 1987 424 pag.
22. Sistemas Operativos MS-DOS, Unix, OS/2, MVS, VMS, OS/400; E. Alcalde, J. Morera, J.A. Pérez Campanero.; Mc Graw Hill; 1992.

### Glosario De Términos En Idioma Inglés

File	File System	mount	Off-line
on-line	Kernel	Volumen Table of Content	End Of File
End Of Tape	Cylinder	Head	Sector
Seek time	Inter Record Gap	Inter Block Gap	Data Field
Key	Record	Directoty	Data Bank
Data Base	map	unmap	Page Fault
Systems Calls	Seek	Write	resource fork
Data fork	Offset	Rename	Append
Copy	Open	Close	Create
Read	Exit	Rewind	Delete
Reset	Sleep	Wakeup	Binding
Status word	Partition table	Boot sector	Fixed Data
Free Space List	File Allocation Table	File Directory Block	Master File Directory
User File Directory	BIOS parameter Block	bit map o bit vector	link
Free Space list head	Preallocation policy	First Fit	Best Fit
Worst Fit	Linked allocation	Index Allocation	Index Block
Header	List Directory	Backup	Device directory
File directory	Hash	Two level directory	path name
search path	Tree structure directory	Acyclic graph directories	Symbolic link
Duplicated entry	word	write next	read next
Sequential Access Method	Password	Root	Boot Block
Super Block	Data Block	Link count	Time stamp s
Changue	File type	Data address	On execution Flags
Last access time	Last modified time	Last changed time	Block size
Major device number	Minor device number	pipes	named pipes
free block	Linked list	Locking	Address holding
overflow	underflow	long words	Last In Last Out

American Standard Code for information interchange

### Glosario De Términos En Castellano

Archivo	Sistema de gestión de Archivos
Tipo de Archivos	Propietario
Soporte de Información	Bloques
Bloqueo	Factor de Bloqueo
Archivo en Cintas	Archivo en Discos
Cilindro	Cabeza
Sector	Bloques de Disco
Registro físico	Registro lógico
Empaquetado	Fragmentación interna
Fragmentación externa	Objetivos del Sistema de Gestión de Archivos
Funciones del Sistema de Gestión de Archivos	Conflictos en un Sistema de Gestión de Archivos
Estructura de la Información	Estructura de un Archivo

Operaciones sobre un Archivo	Catalogación de los Archivos sobre un soporte
Área de datos fijos	Área de Catalogo
Área de datos	Administración del espacio de Almacenamiento
Administración del espacio libre	Administración del espacio ocupado (Asignado)
Compactación o defragmentación	Sistema de directorios
Niveles de directorios	Vínculos simbólicos
Directorio de grafos generalizados	Metodo de acceso
Protección	Métodos de implementación de sistemas de archivos
Sistema de archivos en UNIX	

### Acrónimos Usados En Este Módulo

E/S	Entrada / Salida	Px	Proceso x
DMA	Direct Memory Access	FF	First Fit
PCB	Process Control Block	BF	Best Fit
I/O	Input/ Output	WF	Worst Fit
RAM	Random Access Memory	FAT	File Allocation Table
ROM	Read Only Memory	HD	Head
CPU	Central Processing Unit	VTOC	Volumen Table Of Content
M.C.	Memoria central	EOT	End Of Tape
S.O.	Sistema Operativo	EOF	End of File
r/w/x	Read / Write/Execute	Sec	Sector
Cyl	Cylinder	IRG	Inter Record Gap
VMS	Virtual Machine System	DEC	Digital Equipment Company
SYSCALL	System Call	PCB	Process Control Block
FCT	File Control Table	FET	File Extended Table
PID	Process Identifier	BCD	Bloque de Control del Driver
BCU	Bloque de Control de la Unidad (Dispositivo)	CFP	Current File Pointer
BOF	Begin of File	BOFP	Begin Of File Pointer
EOFP	End Of File Pointer	O.K.	okey
FDA	Fixed Data Area	BPB	BIOS Parameter Block
BIOS	Basic Input Output System	GBy	Giga Byte
FSLH	Free Space List Head	md	make directory
MFD	Master File Directory	UFD	User File Directory
USR	User	ISAM	Index Sequential Access Method
Arch.	Archivo	addr	Address
BS	Block Size	tmp	temporal
ASCII	American Standard Code for Information Interchange		

## ANEXO 7.A. Algoritmos para la Administración de Archivos

A continuación proponemos los siguientes diagramas de System Call para un Sistema de Archivos:

1) CREACION

2) LECTURA:

\*ACCESO SECUENCIAL

\*ACCESO A UN LUGAR ESPECIFICO CON ACCESO SECUENCIAL

\*ACCESO DIRECTO

3) ESCRITURA:

\*ACCESO SECUENCIAL

\*ACCESO DIRECTO

4) OPEN

5) CLOSE

6) ELIMINACION:

\*ARCHIVO

\*DIRECTORIO

7) ASIGNACION ENLAZADA:

\*CREACION (ASIGNACION)

\*ESCRITURA

\*FAT

8) ASIGNACION INDIZADA:

\*CREACION

\*ESCRITURA

9) ASIGNACION CONTIGUA:

\*BUSQUEDA:

-free list space

-mapa de bits

-lista de recuentos

\*ASIGNACION

\*POSIBILIDADES SI EL ARCHIVO MODIFICA SU TAMAÑO

**Algoritmo: CREAR: (ARCHIVO, PARAMETROS)**

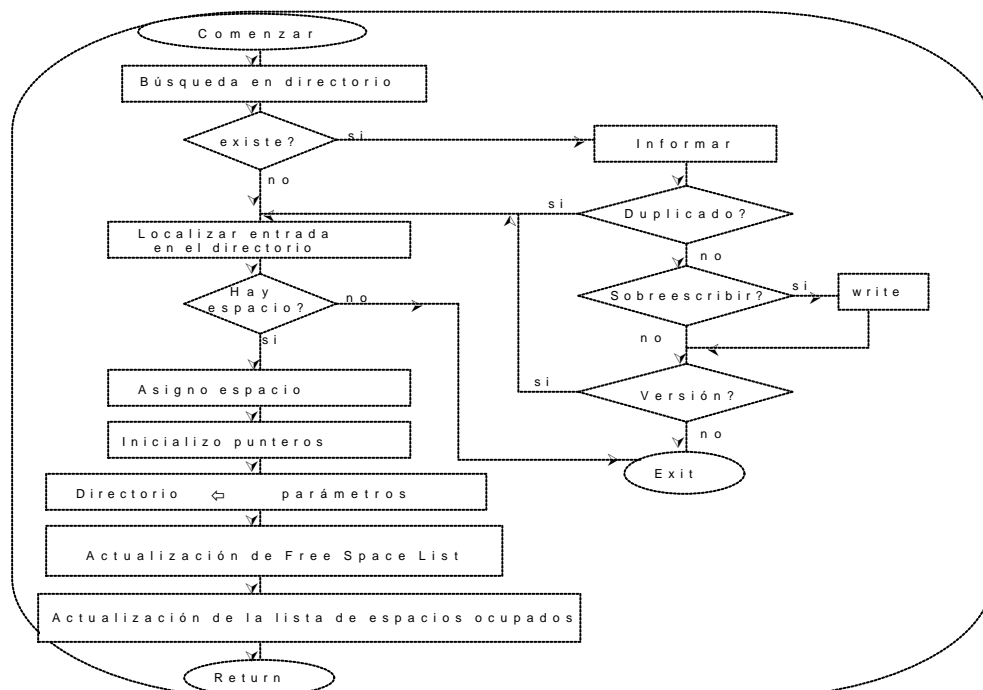


Fig. 7.A.1 Algoritmo para crear un Archivo.

LECTURA:

\*ACCESO SECUENCIAL:

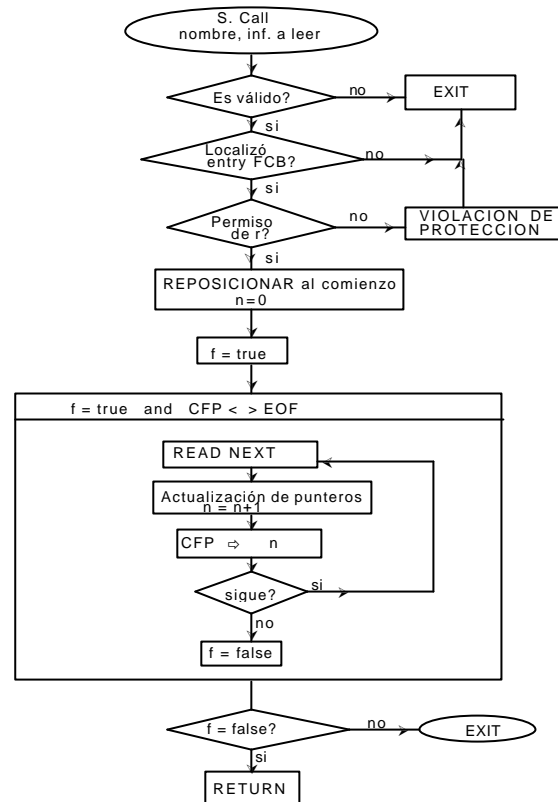


Fig. 7.A.2. Algoritmo para una lectura de un Archivo en acceso secuencial.  
SI SE DESEA LEER UN LUGAR ESPECIFICO: ( Z )

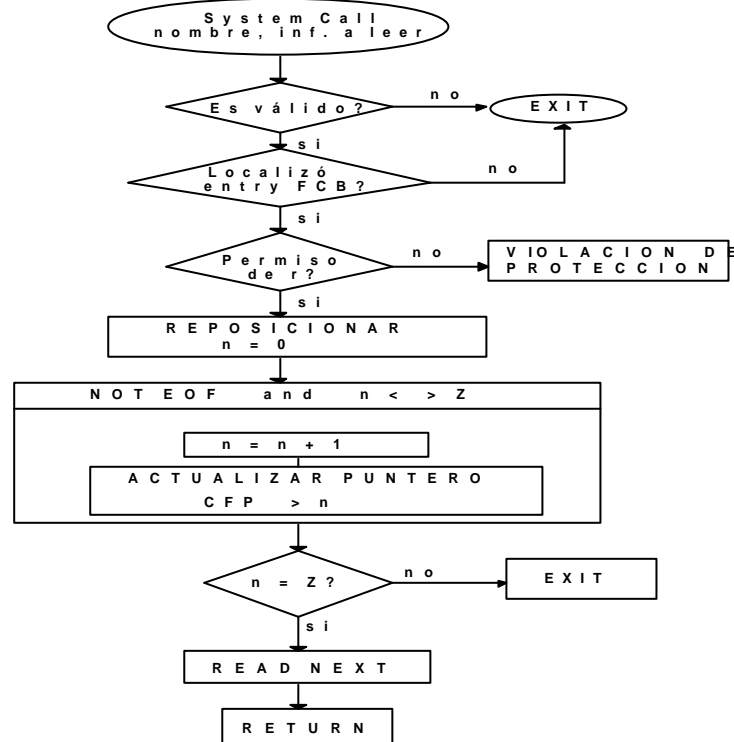


Fig. 7.A.3 Algoritmo para una lectura de un registro específico de un Archivo.

\* ACCESO DIRECTO:

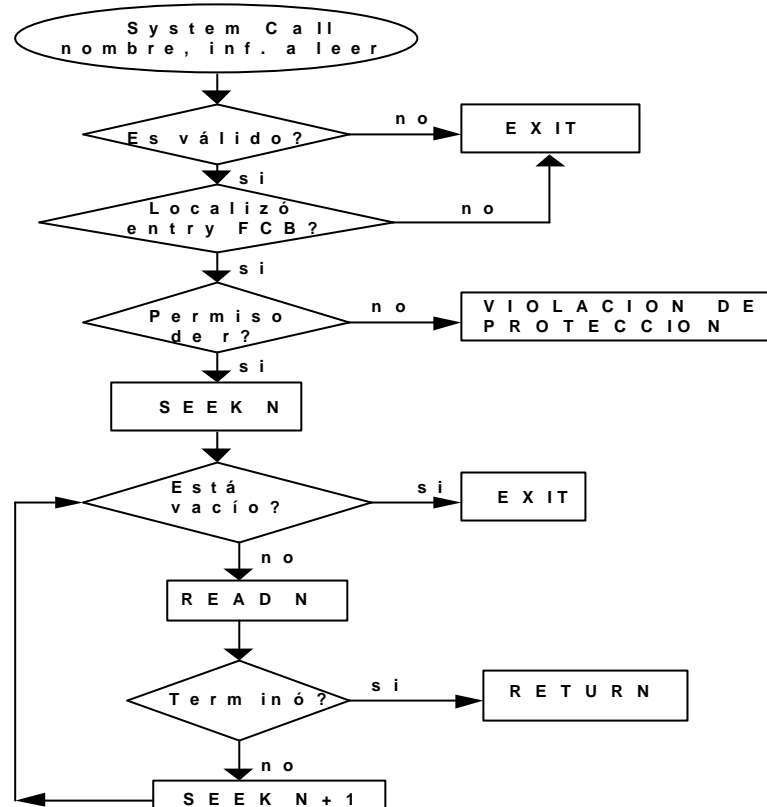


Fig. 7.A.4 Algoritmo para una lectura de un Archivo en acceso Directo.

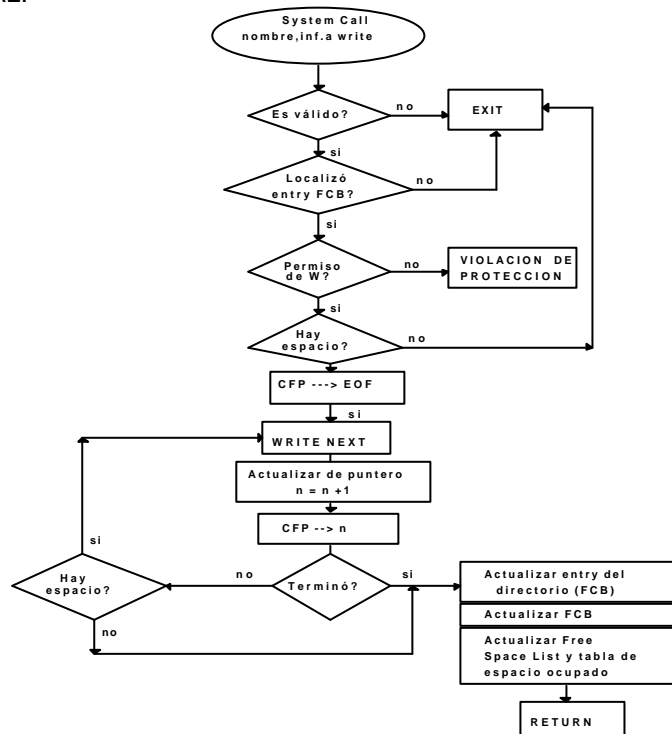
**ESCRITURA:****ACCESO SECUENCIAL:**

Fig. 7.A.5 Algoritmo para una escritura de un Archivo con acceso secuencial.

**ACCESO DIRECTO:**



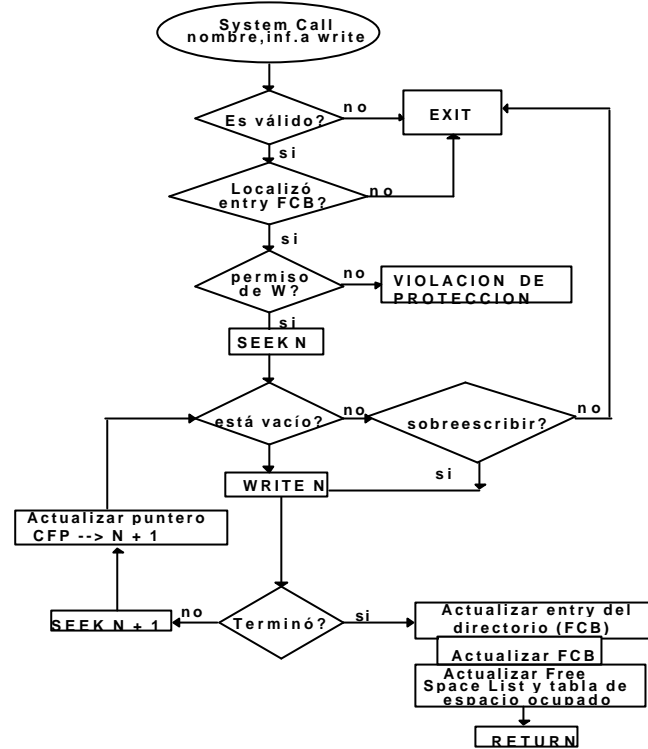


Fig. 7.A.6 Algoritmo para una escritura de un Archivo con acceso directo.

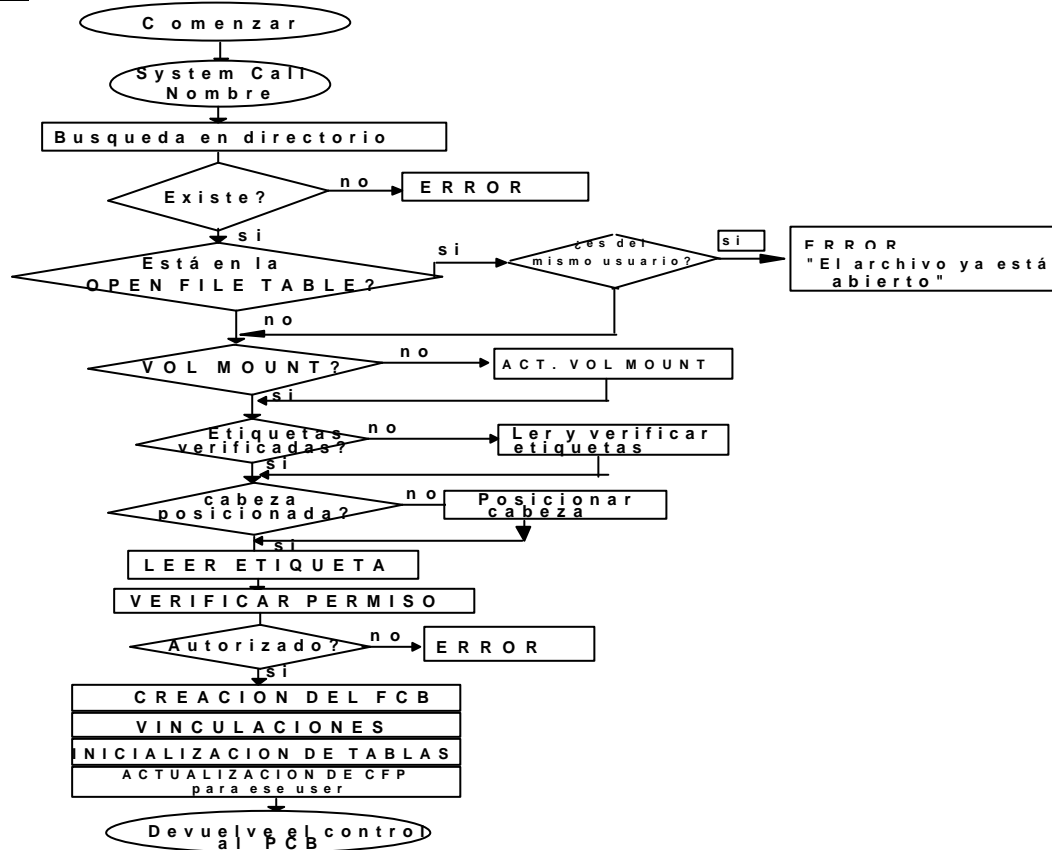
OPEN:

Fig. 7.A.7 Algoritmo para una apertura (open) de un Archivo.

CLOSE:

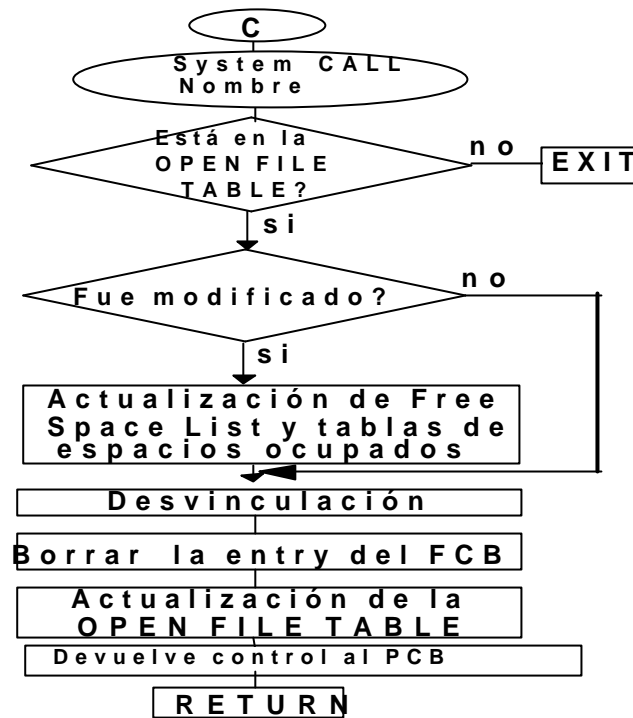


Fig. 7.A.8 Algoritmo para un cierre (close) de un Archivo.

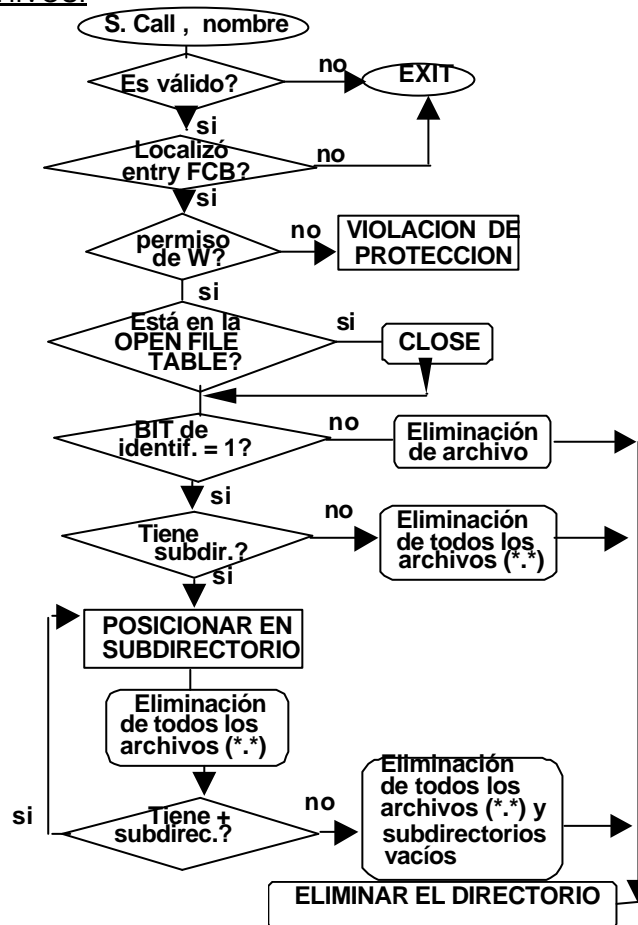
ELIMINACION DE ARCHIVOS:

Fig. 7.A.9 Algoritmo para eliminar Archivos.

CREACION y ELIMINACIÓN de DIRECTORIO: ASIGNACION ENLAZADA

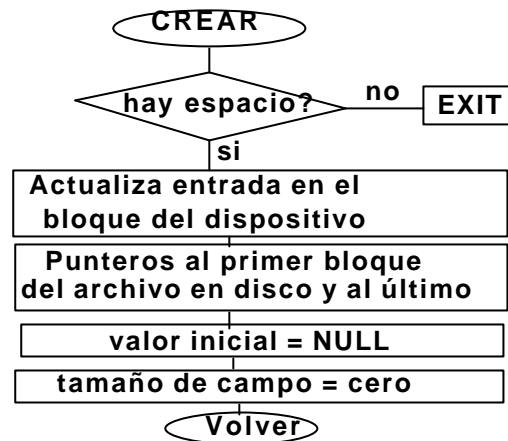


Fig. 7.A.10 Algoritmo para crear una entrada en un Directorio.

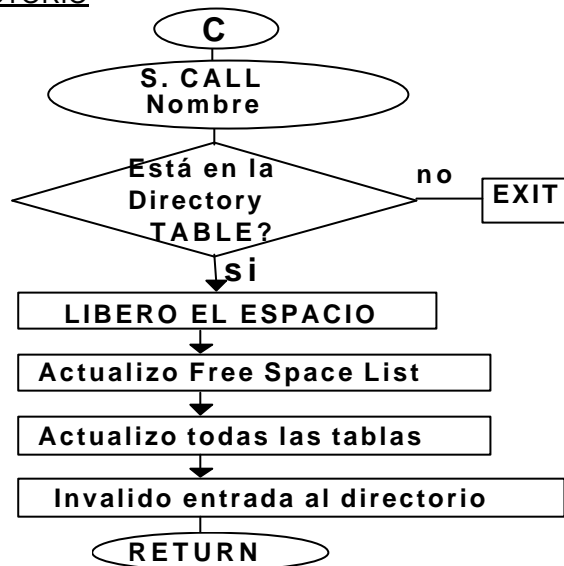
a. ELIMINAR UN DIRECTORIO

Fig. 7.A.11 Algoritmo para eliminar un Directorio.

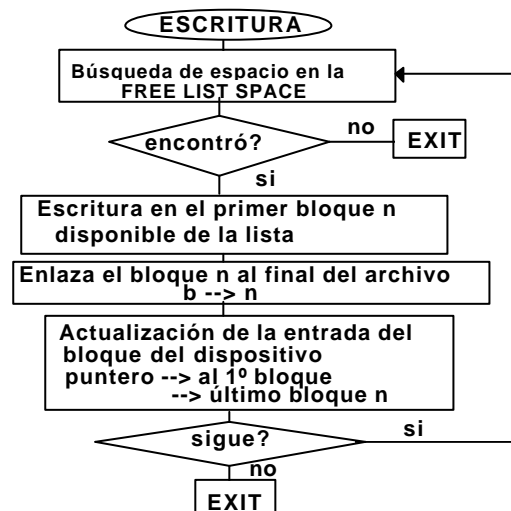
b. ESCRITURA:

Fig. 7.A.12 Algoritmo para crear una entrada en el Directorio.

c) FAT

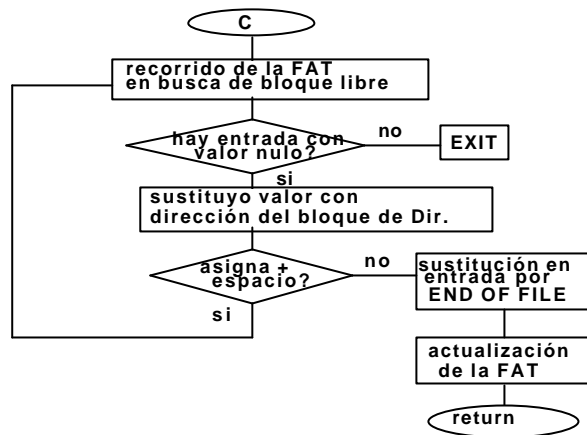


Fig. 7.A.13 Algoritmo para crear una entrada en la FAT.

**ASIGNACION INDIZADA:****A) CREACION**

Fig. 7.A.14 Algoritmo para crear una entrada en el Directorio.

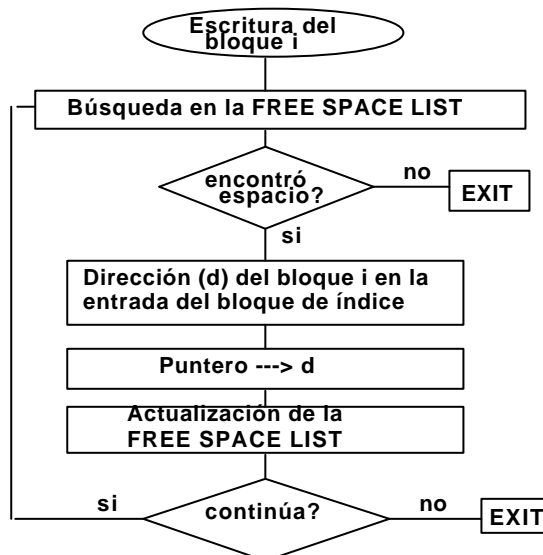
**B) ESCRITURA:**

Fig. 7.A.15 Algoritmo para escribir un bloque de datos.

**ASIGNACION CONTIGUA:****Asignación de n bloques:**

-Búsqueda en lista de espacios libres:

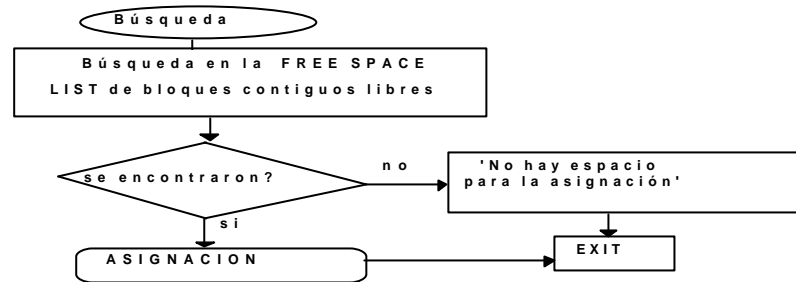


Fig. 7.A.16 Algoritmo para asignar n bloques de datos

■ Búsqueda en mapa de bits:

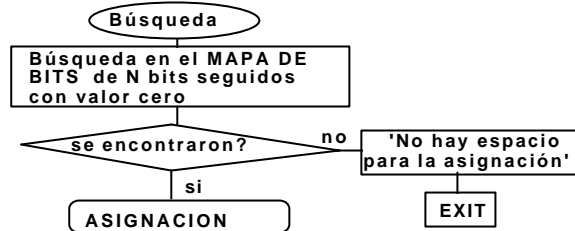


Fig. 7.A.17 Algoritmo para asignar n bloques de datos

- Búsqueda en la lista de direcciones y recuentos:

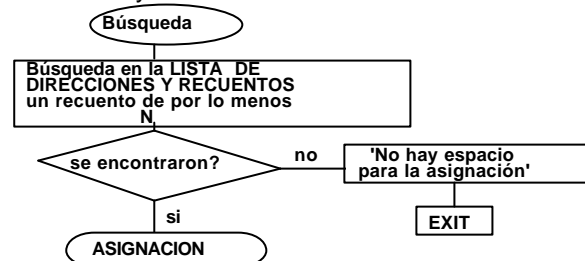


Fig. 7.A.18 Algoritmo para asignar n bloques de datos

-ASIGNACION:

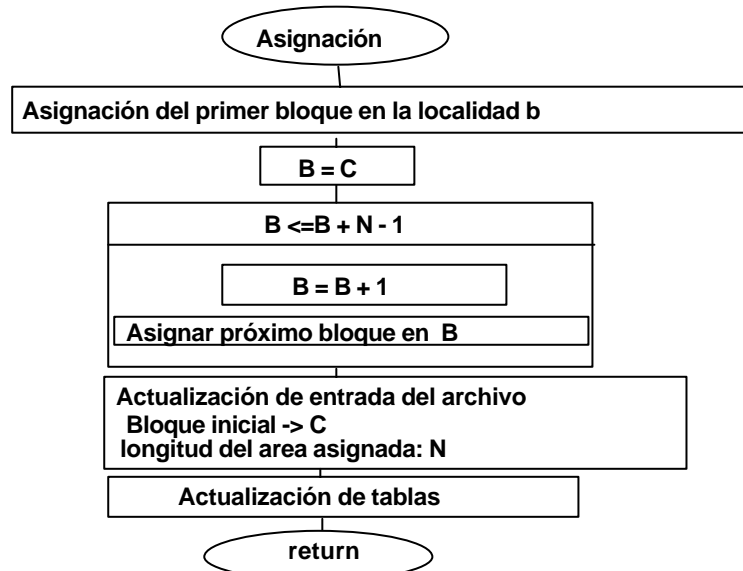


Fig. 7.A.19 Algoritmo para asignar n bloques de datos

-Si se asigna poco espacio o el archivo crece:

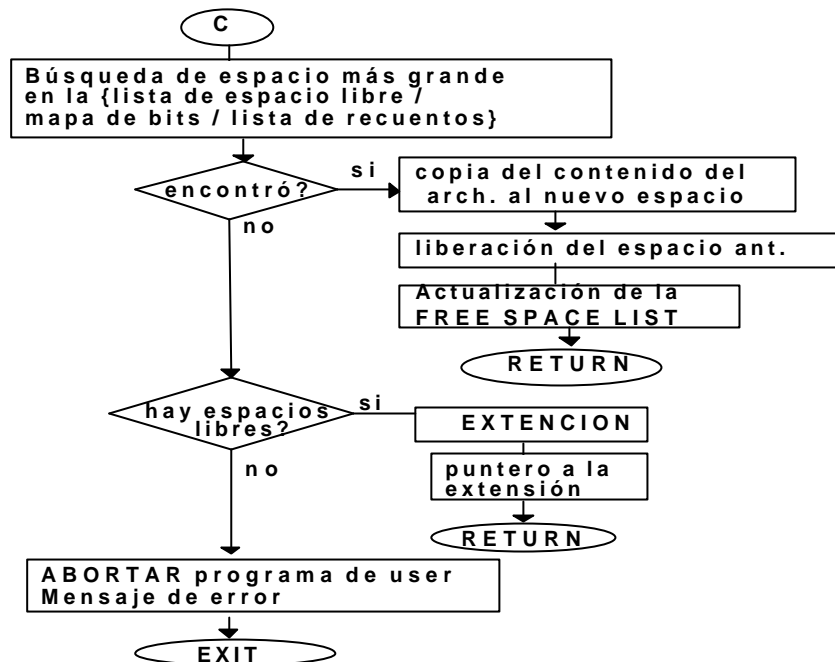


Fig. 7.A.20 Algoritmo para asignar n bloques de datos en un archivo

## ANEXO 7.B FILE SYSTEM EN UNIX

### Objetivos de este Anexo:

Con la finalización de esta unidad, el alumno deberá estar en condiciones para:

- Explicar las razones de un File System
- Describir la estructura lógica del File System
- Describir cada uno de los bloques o áreas especializadas que componen el File System
- Describir la estructura y uso del Super block residente en el disco
- Describir la I-List
- Describir la tabla de montaje.
- entender las relaciones entre las diferentes tablas

### PROPÓSITO DEL FILE SYSTEM

- ESTRUCTURA: provee una estructura para la colección de archivos en el sistema.
- IDENTIFICACIÓN: provee los medios para la identificación única en el sistema.
- SEPARACIÓN LÓGICA: provee los medios para la separación lógica de entidades (archivos) por seguridad y backup de los datos.
- SEPARACIÓN FÍSICA: provee los medios para la separación física de entidades (archivos) por seguridad y backup de datos.
- ESTRUCTURA JERÁRQUICA: provee medios para reestructuración jerárquica en UNIX System File.

### ARCHIVOS EN UNIX

En UNIX la organización de los archivos es jerárquica, organizada en directorios, presentando una estructura en forma de árbol, en donde los directorios son ramas y los archivos hojas. El directorio raíz (/) es el directorio principal, del que cuelgan subdirectorios, de los que a su vez cuelgan otros directorios o subdirectorios, etc.. En principio se puede suponer que no existe limitación en cuanto a amplitud como profundidad de la estructura, pero en realidad si existe dicha limitación, dependiendo esta de la versión, del espacio en disco disponible, etc.

Un sistema de archivos en UNIX es necesario montarla (hacerla accesible). Montar un sistema de archivos consiste en asignar un directorio, o punto de montaje, a la unidad lógica.

El montaje de un sistema de archivos se realiza con el comando **mount**, y el desmontaje con el comando **umount**. Ambas tareas solo son realizables, en principio, por root.

mount Dispositivo Directorio	Monta el Dispositivo en el directorio especificado.
mount	Muestra los dispositivos montados.
Umount Directorio	Desmonta el Directorio especificado.

Una unidad lógica en UNIX si puede extenderse a varias unidades físicas, si bien esto depende de la versión de UNIX instalada.

En UNIX cualquier dispositivo se trata como un archivo. Un terminal tiene asociado un archivo, y por tanto si se escribe algo en ese archivo, aparecerá por el terminal. Lo mismo sucede con las impresoras, módem, etc..

/dev/rmt/0m	Primera unidad de cinta.
/dev/floppy	Unidad de disquete.
/dev/ttyXX	Terminales.

Para acceder a un directorio se puede utilizar el path o camino relativo o absoluto. El path absoluto se referencia al directorio raíz, por lo que siempre comenzará por el carácter "/" indicativo del directorio raíz. El path relativo, en cambio, se referencia al directorio en que se encuentra el usuario en ese momento (comenzará por "." si se refiere al directorio superior al actual, o por "." o el nombre de un subdirectorio si se refiere al un subdirectorio del directorio actual).

cd /usr/bin	Path absoluto
cd ../bin	Path relativo

En UNIX los archivos tienen permisos para el usuario, para el grupo del usuario y para el resto de usuarios. Con esto se puede hacer que un archivo sea accesible para un determinado usuario, o grupo de usuarios, que solo pueda modificarlo un usuario, etc. Los permisos de un archivo se indican con 10 caracteres se asignan con números:

Al ejecutar el comando "**ls -l**" nos aparecerá al principio de cada línea una información del tipo:

-rwxrwxrwx	usuario	grupo	...
------------	---------	-------	-----

El primer carácter hace referencia al tipo de archivo. El primer grupo rwx hace referencia al usuario propietario del archivo. El segundo grupo corresponde a los usuarios que pertenecen al mismo grupo que el propietario. El tercero pertenece al resto de usuarios. Root tiene acceso ilimitado a todos los archivos, aunque no tengan activado ningún permiso. Si aparece la letra indica que está permitido ese permiso, y si aparece un guión indica que está prohibido.

- r** : acceso a lectura (4).
- w** : acceso de escritura (2).
- x** : acceso de ejecución (1).
- : sin permiso (0).

Para cambiar los permisos de un archivo se utiliza el comando **chmod** (change mode):

chmod 750 archivo o chmod u=rwx g=rx o=-rwx archivo

Dará permiso de lectura, escritura y ejecución para el propietario, de lectura y ejecución para el grupo de usuarios al que pertenezca el propietario, y ningún permiso para el resto de usuarios.

Para cambiar el propietario de un archivo se utiliza el comando **chown** (change owner) y para cambiar el grupo del usuario **chgrp** (change group):

chown fulanito[:grupo] archivo	hace que fulanito pase a ser el propietario de archivo. Se puede especificar también el grupo.
chgrp users archivo	hace que el grupo users sea considerado como grupo del propietario

Existen unos permisos especiales que son:

**s** (Bit s): Hace que cualquier usuario que ejecute el programa adquiera la identidad del propietario durante la ejecución. (4000). También se puede hacer que el usuario que lo ejecute pase a ser del grupo del propietario durante la ejecución (2000).

**t** (Sticky bit): Hace que el programa se lea la primera vez de disco, y quede residente en memoria, con lo que la próxima vez que se ejecute se cargará más rápidamente (1000).

Hay 4 tipos básicos de archivos en UNIX:

- Regular: contiene datos.
- Directorio: Archivo de estructura especial.
- Especial (Dispositivos): Ejemplos /dev.

- Llamados Pipes

NOTA: Dos tipos de archivos adicionales: multiplexed character device y multiplexed block device son reservados para uso futuro y corrientemente no se mencionan.

### **ARCHIVOS REGULARES:**

Los archivos se especifican por:

**{camino jerárquico}/nombre{.ext}**

donde las llaves ({} ) pueden ser o no necesarias. Los archivos pueden constar de una extensión que es lo que aparece tras el punto (.). Un ejemplo sería:

/usr/lib/starbase/demos/star.f

Entre los sistemas de archivos más comunes tenemos:

**/** : Sistema de archivos raíz

**/home** : Sistema de archivos para ubicación de los directorios de usuario.

**/tmp** : Sistema de archivos para temporales. Puede estar ubicado en memoria RAM (disco RAM), con lo que el acceso será más rápido.

**/usr** : Sistema de archivos para archivos ejecutables, documentación, referencia.

**/var** : Sistema de archivos para logs, auxiliares, archivos que crecen.

- Archivos regulares contienen diversos datos que el usuario decide introducir en él.

- La estructura de los archivos regulares es controlada por el programa Usuario. El S.O. no se responsabiliza sobre la estructura.

Datos

Estructura impuesta por el usuario

Lectura

Escritura

} De acuerdo al permiso de acceso.

- Ejemplos: /etc/passwd (ASCII)

/bin/ps (binario)

/usr/src/cmd/lsc (C-Source)

### **ARCHIVOS DIRECTORIOS**

El kernel de UNIX ve a todos los archivos como cadenas de bytes. Los archivos en UNIX se organizan en **directorios**; también se organizan de esta forma como todos los dispositivos accesibles desde UNIX, como impresoras, pantalla y discos. Un directorio es como una carpeta que contiene archivos y otros directorios, de forma que toda la estructura de directorios se organiza como un árbol, con los archivos como hojas de las ramas del árbol.

A la vez, todos los archivos en UNIX descienden de un directorio principal, /, o **directorio raíz**; de este archivo descienden todos los archivos de todas las unidades: discos duros, CD-ROM y disquetes, por ejemplo. Los directorios se separan con el mismo símbolo, /. En UNIX la unidad va implícita en el nombre del archivo (por ejemplo, del directorio /disk02 cuelgan todos los archivos que se hallan físicamente en ese disco). Y dado que UNIX es un sistema operativo de red también, del directorio raíz cuelgan también las unidades remotas, o situadas en otros ordenadores; el acceso a sistemas de archivos de otros ordenadores se hace de forma transparente al usuario, simplemente cambiando de directorio.

Cada archivo tiene un nombre único dentro de cada sistema, y este nombre único está compuesto por todos los directorios que hay que recorrer hasta llegar a él. Este se denomina el **camino**; por ejemplo un archivo que estuviera dentro del directorio "toto" en la siguiente figura, tendría el camino /usr/toto/archivo.

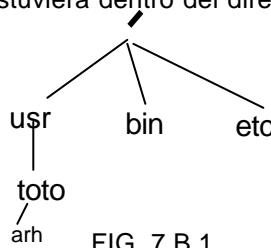


FIG. 7.B.1

En UNIX, se distinguen mayúsculas y minúsculas en los nombres de archivo, y además pueden tener hasta 256 caracteres de largo, incluyendo espacios, asteriscos, y otros caracteres.

Dentro de cada directorio hay siempre dos archivos: "." y "..", directorio actual, y directorio superior. Pueden usarse como atajos para referirnos a alguno de esos directorios. Porque nadie querría hacer eso, ese es otro tema. Además, el directorio ~ es el directorio base del propio usuario.

- Archivos directorio contienen datos que proveen un mapeado de nombres de archivos sobre el correspondiente soporte.



- La estructura de los archivos directorios son leíbles por el programa del usuario y son grabados solamente por el S.O.

Datos: Nombres  $\Rightarrow$  i-nodos

Estructura: S.O. Standard

Lectura: Programa usuario

Escritura: S.O.

- Ejemplos:

/ (root = directorio raíz)

/dev (device directory = directorio de dispositivos)

/bin (command directory = directorio de comandos)

Al elegir los nombres de los archivos, es conveniente limitarse a utilizar sólo los caracteres que correspondan a letras, números, el carácter subrayado \_ y el carácter punto. Los archivos cuyo nombre comience por punto permanecen ocultos.

En UNIX existe una jerarquía de directorios que para un sistema estándar sería:

En UNIX existe una estructura de directorios que difieren poco de un sistema a otro. Entre los directorios más comunes tenemos:

**/** : directorio raíz. De él cuelgan todos los demás directorios.

**/bin** : archivos ejecutables, comandos de usuario.

**/cdrom** : punto de montaje de CD.

**/dev** : archivos de dispositivos (discos, terminales, etc.).

**/etc** : archivos de configuración, administración e información del sistema.

**/floppy** : punto de montaje de disquetes.

**/home** : archivos de usuarios.

**/lib** : archivos de bibliotecas de desarrollo y material de apoyo.

**/lost+found** : archivos perdidos.

**/mnt** : punto de montaje de dispositivos externos.

**/sbin** : archivos ejecutables de administración.

**/tmp** : Archivos temporales o zona de trabajo de algunos programas UNIX.

**/usr** : archivos ejecutables, documentación, referencia.

**/var** : archivos log y auxiliares.

El directorio raíz es el único que no tiene nombre. Cada usuario tiene un directorio **home** que es el directorio asignado a ese usuario para que almacene sus archivos. El camino de este directorio está contenido en la variable HOME.

Los archivos pueden tener cualquier longitud hasta 256 caracteres. Como se ha dicho anteriormente, la extensión es opcional. En caso de que exista, por convenio significa:

- .f Programa fuente escrito en lenguaje FORTRAN
- .p Programa fuente escrito en lenguaje PASCAL
- .c Programa fuente escrito en lenguaje C
- .o Archivo objeto
- .a Biblioteca de módulos
- .h Archivo de "cabecera"

Un archivo puede ser referenciado por su nombre precedido del nombre del directorio que le contiene. En cualquier momento, los comandos que se teclean al shell están dados en referencia al directorio actual, es decir, el directorio en el que nos "encontramos". Cuando nos referimos a un archivo lo podemos hacer en referencia al directorio actual (vía de acceso relativo) o en referencia al directorio raíz (vía de acceso absoluto). En la vía de acceso relativo podemos hacer uso de la notación que significa una referencia al directorio padre. Para clarificar todo esto, pongamos un ejemplo:

Supongamos la siguiente estructura de archivos-directorios:

```

/users/pepe                               Informa.txt
      /mis_archivos                       Trabajo.f
      Toto.c
      /otros_archivos                     Toto.o
      Tato.a
      /citas                             Carmen
      Juana

```

y que nos encontramos en el directorio otros\_archivos. Las referencias a los diferentes archivos se harían de la siguiente manera:

Nombre del archivo	Acceso absoluto	Acceso relativo
Informa.txt	/users/pepe/informa.txt	../informa.txt

Trabajo.f	/users/pepe/mis_archivos/trabajo.f	../mis_archivos/trabajo.f
Toto.c	/users/pepe/mis_archivos/toto.c	../mis_archivos/toto.c
Toto.o	/users/pepe/mis_archivos/toto.o	../mis_archivos/toto.o
Tato.a	/users/pepe/otros_archivos/tato.a	../tato.a
Carmen	/users/pepe/otros_archivos/citas/carmen	../citas/carmen
Juana	/users/pepe/otros_archivos/citas/juana	../citas/juana

Normalmente, será más corto y sencillo utilizar el acceso relativo excepto en determinados casos (por ejemplo, en nuestro caso, /bin/fgrep).

UNIX no mantiene versiones de archivos, por lo que es necesario prestar especial atención a acciones como borrarlos o modificarlos.

Existen 3 archivos estándar implementados en UNIX:

- Entrada estándar (stdin): Teclado (0)
- Salida estándar (stdout): Pantalla (1)
- Errores estándar (stderr): Pantalla (2)

UNIX emplea un sistema de archivos jerárquico de directorios-archivos.

No existe, en el nivel de usuario, el concepto de *volumen*, ni de *dispositivo físico*. Es decir, el usuario no sabe en qué disco están los archivos que está utilizando.

Un archivo es un conjunto de información al que se le da un nombre (nombre del archivo).

### ARCHIVOS ESPECIALES

- En los archivos especiales los datos son almacenados en y transmitidos por los dispositivos que representan.
- Los archivos especiales proveen un mapa de los nombres de archivos en UNIX soportados en dispositivos.
- Archivos especiales representan caracteres, block, bloques de línea y terminal de dispositivos de Entrada/Salida.

Datos: nombre  $\Rightarrow$  mayor, menor número de dispositivos (tipo)

Estructura: Inodos

Lectura: }

Escritura: } Permiso de acceso del usuario

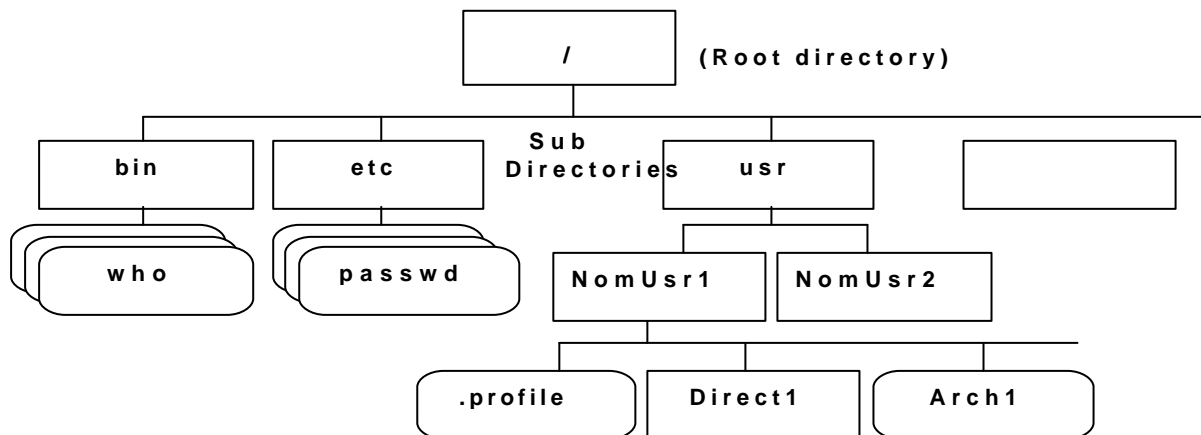
- Ejemplos: /dev/tty (terminal device)

/dev/dsk (disk block device)

/dev/aeu (auto call unit)

### LLAMADOS PIPES

- Los llamados pipes contienen cualquier dato que el programa del usuario coloca en él. Estos datos no pueden ser mayores que 10 blocks (capacidad máxima).
- La escritura de los llamados pipes es controlada por el programa del usuario.
- La lectura y escritura de los llamados pipes son realizados por el programa usuario asumiendo su propio permiso de acceso.



Los rectángulos del gráfico representan directorios, y los rectángulos con bordes redondeados representan archivos.

Fig. 7.B.2 estructura jerárquica de archivos en UNIX

- Datos: ]
- Estructura: ] Determinado por el usuario.
- Lectura: ]
- Escritura: ] Permiso de Acceso del usuario.

### ESTRUCTURA LÓGICA DE ARCHIVOS EN UNIX

■ La estructura lógica de archivos en UNIX se describe mejor mediante un "árbol invertido" (jerárquico)

El File System comienza con la Raíz = root (/) que es un directorio especial de comienzo / finalización.

Se puede acceder a cada archivo del "árbol" al proveer un camino relativo o pleno (pathname).

La forma en que los archivos están físicamente separados en el S.O. UNIX no es obvio para el usuario.

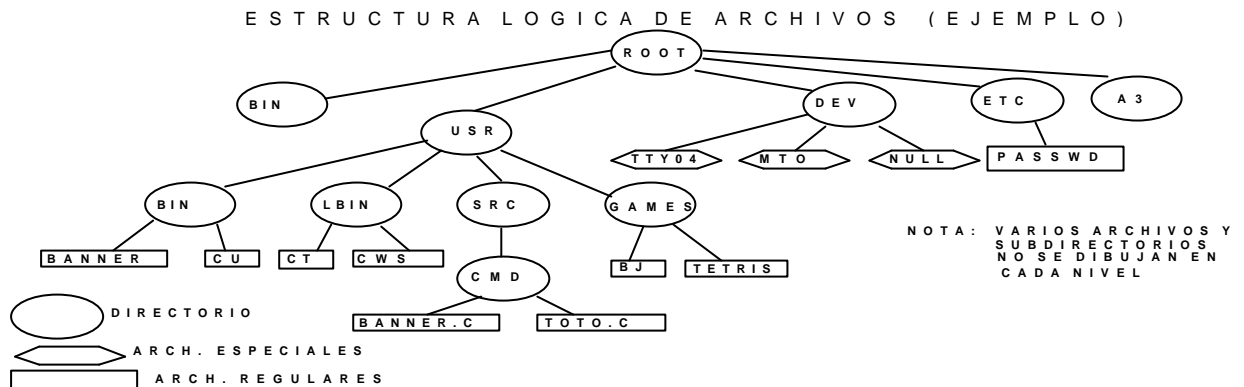


Fig. 7.B.3 Estructura lógica de archivo en UNIX

### ARCHIVOS EN UNIX (ISAM)

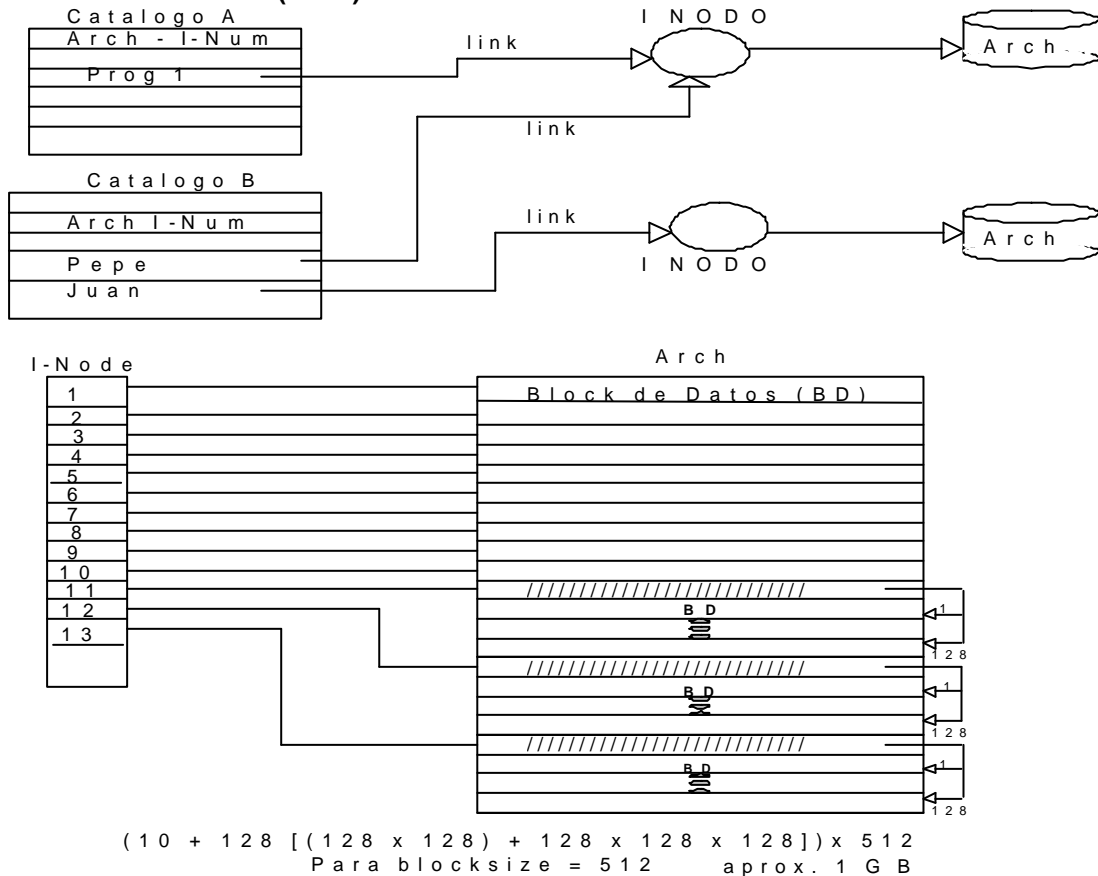


Fig. 7.B.4 Estructura ISAM en UNIX.

### ASIGNACION DE ARCHIVOS:

La asignación de archivos se hace en un bloque. Esta asignación es dinámica. Y los bloques no tiene

que ser necesariamente contiguos. Un método de indexado es utilizado para mantener la información de cada archivo, en donde una parte del índice es almacenado en un inode. Los inodes incluyen una dirección de información de 39 bytes, organizada como 13 direcciones o punteros de 3 bytes cada una. Las primeras 10 direcciones apuntan a los 10 primeros bloques de datos del archivo.

### **INODOS (INODES)**

Todos los tipos de archivos de UNIX son administrados por el sistema operativo mediante los llamados INODES, que son estructuras de control que contienen la clave que necesita el sistema operativo para un archivo en particular.

Muchos archivos pueden ser asociados con un i-nodo en particular, pero no sucede al revés, un i-nodo activo es asociado exactamente con un solo archivo, y cada archivo es controlado por un i-nodo.

En un i-nodo lo que se guarda son los permisos de cada archivo.

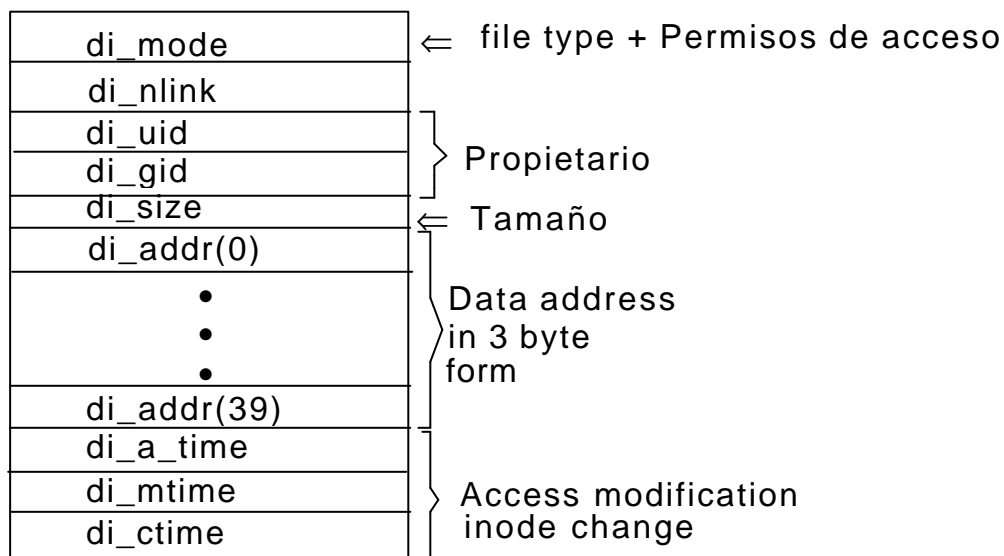
- Tiene una Asociación con los archivos
- Contiene Información sobre atributos de los archivos

Dos formas: Disk o In-core (en memoria central)

- Varios nombres de archivos son asociados con un simple i-nodo (nodo de información) pero un i-nodo es asociado con un (y sólo un) archivo.
- El S.O. usa un i-nodo para almacenar atributos del archivo.
- Un i-nodo está residente en disco, o residente en la tabla de i-nodos del S.O. o en ambos.

### **ESTRUCTURA DE INODOS RESIDENTE EN DISCO**

- Modo (Tipo de archivo + Permisos)
  - Contador de enlace (Link count)
  - Propietario
  - Tamaño
  - Marca de tiempo (Time stamps)
  - Información de direccionamiento
- La estructura de inodo residente en disco está definida en /usr/include/sys/ino.h
- Cada campo de la estructura de inodo residente en el disco comienza con los caracteres di- (disk inode)



Ref: /usr/include/sys/ino.h

Fig. 7.B.5 Estructura de Inodo residente en disco

### **MODO DE INODOS**

- Nombre del miembro di\_mode
- Tipo de archivo (Por ejemplo regular, especial) 4 bits
- "On Execution" flags (Por ejemplo /setuid on exec) 3 bits
- Permiso de Accesos (Por ejemplo rwx)

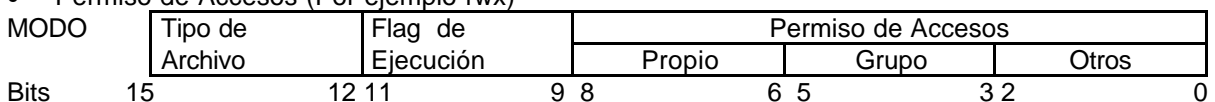


Fig. 7.B.6 Subcampos de Modo en el I-nodo

**INODE LINK COUNT (Contador de enlace de inodos)**

- Nombre del campo miembro di\_nlink
- Valores
- Un contador de enlace de i-nodo contiene:
  - El total de número de referencias de directorios del inodo
  - Un cero ("0") para un inodo no alocado
  - El último 1 para un archivo
  - El último 2 para un directorio

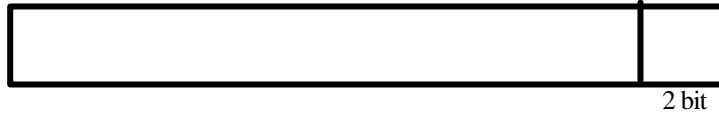


Fig. 7.B.7 contador de enlaces

**PROPIETARIOS DEL I-NODO**

- Dos campos contienen información sobre propietarios
  - Propio del usuario (di\_uid)
  - Miembros del grupo del propio usuario (di\_gid)
  - Valores
- Ambos campos contienen valores enteros que reflejan la información del archivo del propietario y del grupo.

**TAMAÑO DEL INODO**

- Nombres de los miembros (di\_size)
- Usos (tamaño (bytes) de Datos)
- Valores (Regular-directorio)
- Un campo de tamaño de inodo (di\_size) contiene un valor entero igualando el número de bytes de datos en un archivo directorio o regular.

**MARCAS DE TIEMPO DEL INODO**

- Acceso última vez (di\_atime)
- Última vez de modificación (di\_mtime)
- Último cambio (di\_ctime)
- Existen tres campos para anotar las marcas de tiempo en el i-nodo residente en el disco:
  - LAST ACCESS TIME (di\_atime) - La última fecha en que fue leído.
  - LAST MODIFIED TIME (di\_mtime) - La última fecha en que fue modificado.
  - LAST CHANGED TIME (di\_ctime) - La última vez que el i-nodo fue modificado.
- Los tres campos contienen un valor entero que refleja el número de segundos desde las 00:00:00 hs del 01/01/1970.
- Las marcas de tiempo residen solamente en los inodos residentes en el disco.

**INFORMACIÓN DE DIRECCIÓN DEL INODO (disco)**

- Nombres de los miembros di\_addr[0] hasta di\_addr[39]
- Valores dependientes del tipo de archivo
- El campo de dirección del inodo (di\_addr[40]) contiene diferentes tipos de datos, dependiendo del tipo de Archivo del inodo

**CAMPOS DE DIRECCIONAMIENTO DE BLOQUES DE INODO**

Campo de direccionamiento de archivos regulares y directorios directo.

- El campo de direccionamiento de un i-nodo de un archivo regular o de un directorio está dividido en trece subcampos de tres bytes, cada uno contiene bloques de direcciones de dispositivos (Por ejemplo disco)

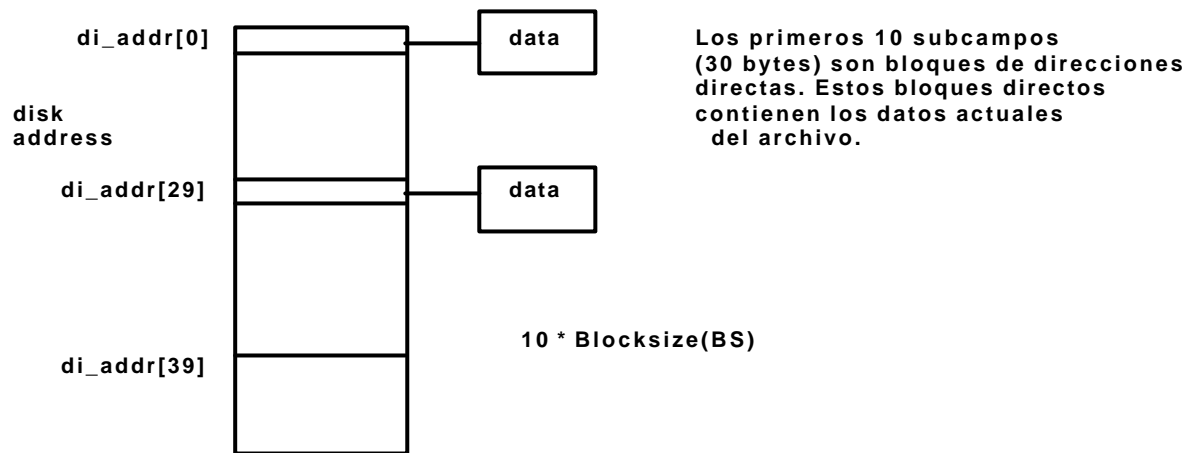


Fig.7.B.8 Subcampos: Direct Block Address.

**Campo de direccionamiento de archivos regulares y directorios indirecto simple.**

- El subcampo 11 es un bloque de direccionamiento indirecto que apunta a un block que contiene bloques de direcciones directas ( $\text{Blocksize} / 4$ )
- El tamaño de bloque ( $\text{Blocksize}$ ) puede tener 512 By o 1024 By.

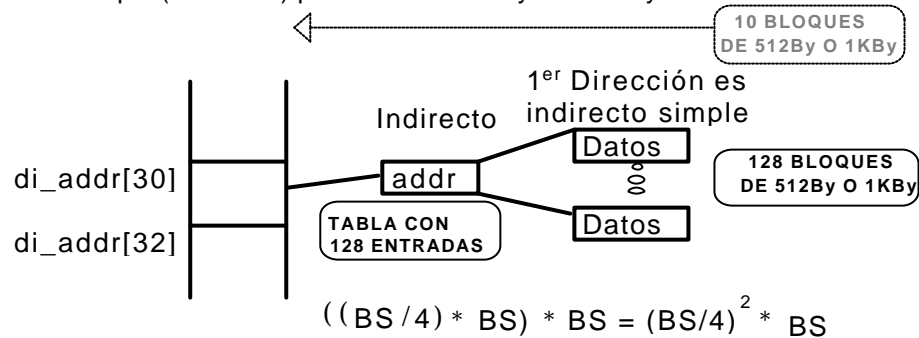
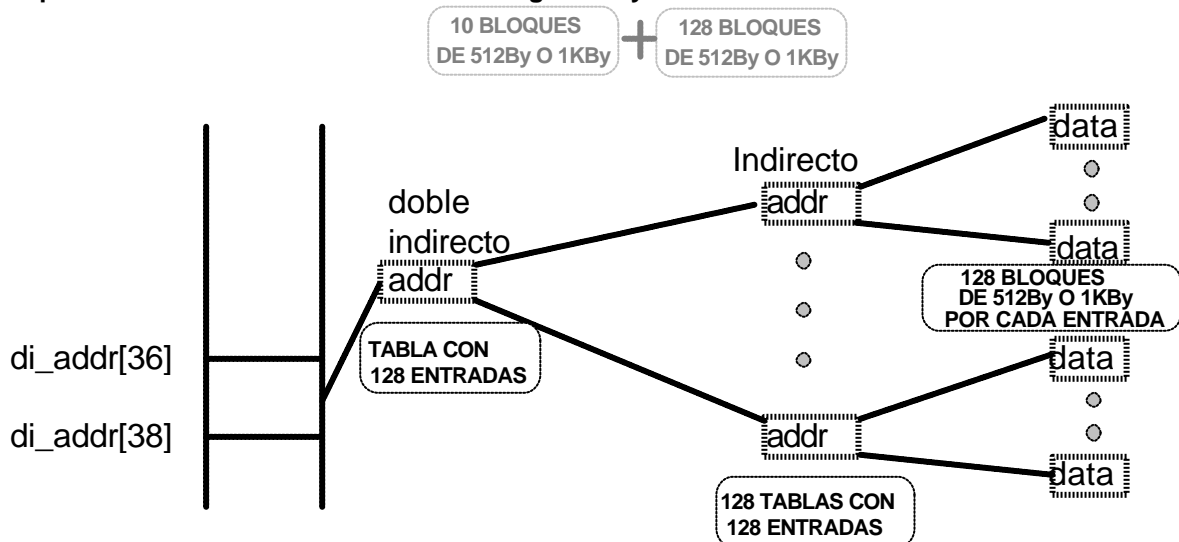


Fig. 7.B.9 Campo de direccionamiento indirecto simple

**Campo de direccionamiento de archivos regulares y directorios doble indirecto.**

$$\text{Size} = (BS) * (BS/4)^2 = 838 \text{ kBy}$$

Fig. 7.B.10 Campo de direccionamiento doble indirecto

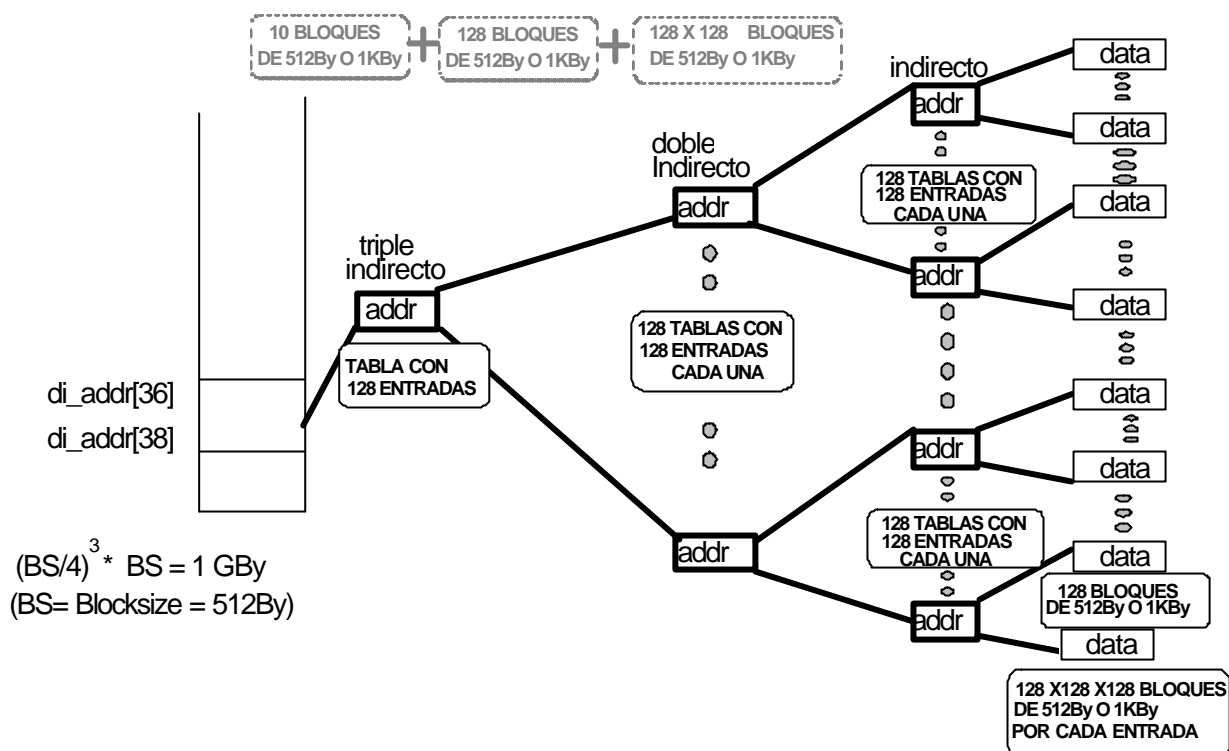


Fig. 7.B.11 Campo de direccionamiento triple indirecto

- El subcampo doceavo es un bloque de direccionamiento triplemente indirecto que apunta a un bloque que contiene bloques de direcciones indirectas dobles (Blocksize / 4)

## Campo de direccionamiento de archivos regulares y directorios triple indirecto

- El subcampo treceavo es un bloque de direccionamiento triplemente indirecto que apunta a un bloque que contiene bloques de direcciones indirectas dobles ( $\text{Blocksize} / 4$ )

## Capacidad de un archivo UNIX

Nivel	Numero de bloques	Numero de bytes
Directo	10	10 K
Simple indirecto	256	256 K
Doble indirecto	$256 \times 256 = 65 \text{ K}$	65 M
Triple indirecto	$256 \times 65 \text{ K} = 16 \text{ M}$	16 G

Esto muestra que el número total de los bloques de datos en un archivo depende de la capacidad del tamaño de los bloques en el sistema. En la tabla se utilizó un bloque de 1 K.

**Campo de direccionamiento de archivos especiales:**

Solamente los primeros dos bytes del campo de direccionamiento son usados.

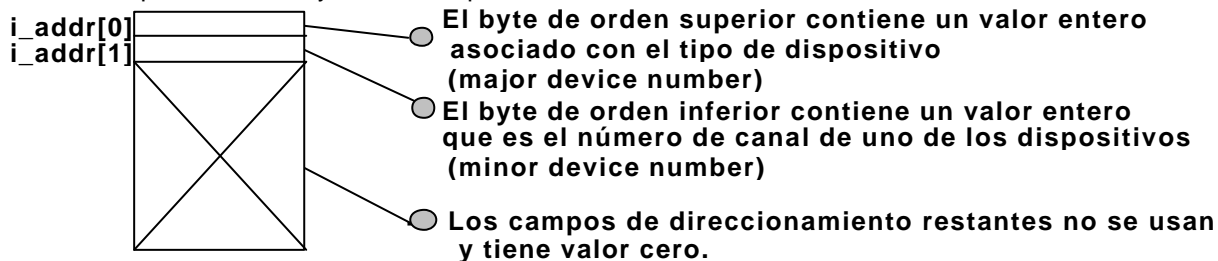


Fig. 7.B.12 Campo de direccionamiento de archivos especiales

## Campo de direccionamiento - Llamadas pipes

- di\_addr[0] hasta di\_addr[29] son usados para contener bloques de direcciones.
- di\_addr[30] hasta di\_addr[39] son usados para Lectura/Escritura, conteo y desplazamientos

NOTA: Llamadas pipes no debe confundirse con regular (no nominados) pipes.

No nominados pipes no tienen correspondencia con i-nodo de disco.

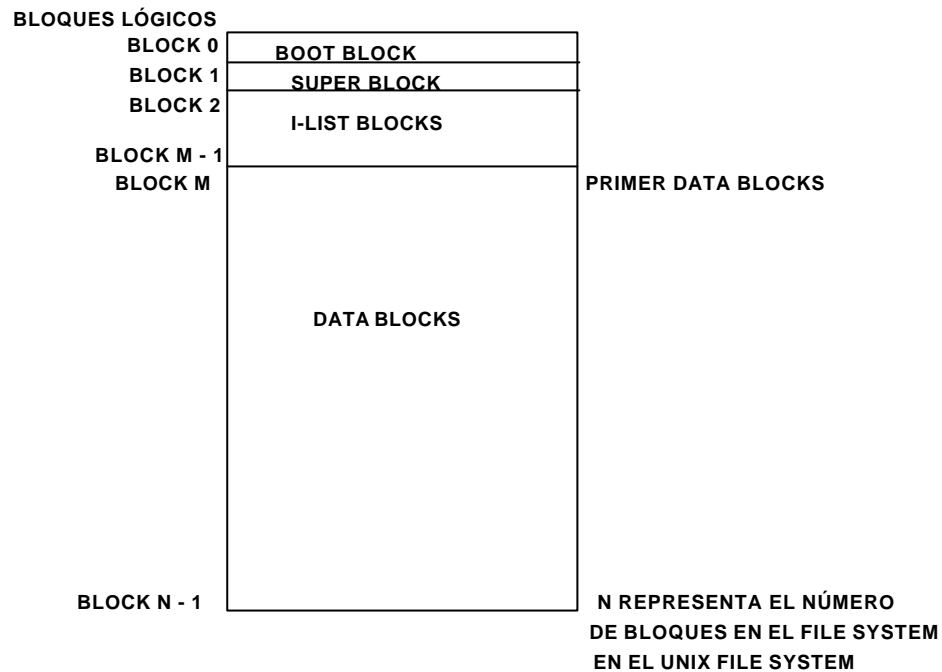


Fig. 7.B.13 Estructura física del File System en UNIX.

**ESTRUCTURA FÍSICA DEL FILE SYSTEM**

Un File System es una colección de Bloques contiguos en un dispositivo de almacenamiento. Cada File System debe contener los siguientes componentes:

Boot Block  
 Super Block  
 I-List Blocks  
 Data Blocks

**ESTRUCTURA FÍSICA DEL BOOT BLOCK**

- El primer block del File System (Block 0) es el Boot Block.
- Muchas veces el Block 0 contiene el programa de arranque (booteo) con el cual se inicializa el UNIX.
- Si la inicialización del UNIX (booteo) no fue realizado con el Block 0, sino con el programa de arranque de otro lugar, entonces el contenido del Block 0 no está definido porque no es usado.

**ESTRUCTURA FÍSICA DEL SUPER BLOCK**

- Los datos específicos del File System y sobre él, son almacenados en el Super Block.
- El Super Block es el segundo Block (Block 1) en el File System.
- El Offset del Super Block es de 512 By.

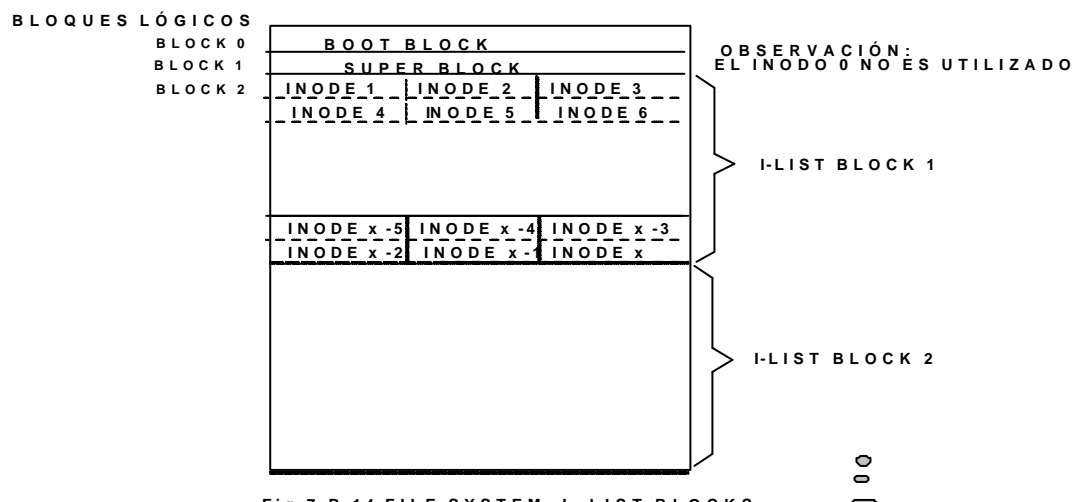
**ESTRUCTURA FÍSICA DE LOS I-LIST BLOCKS**

Fig. 7.B.14 FILE SYSTEM I - LIST BLOCKS

- Un I-nodo por Archivo. Cada Archivo tiene un i-nodo que esta almacenado en el File System.



- Varios inodos puede incluir el File System Block (hasta un máximo de 99).
- Los Bloques que contienen los i-nodos están contiguos pero están separados en la sección de la I-List del File System.
- Los Bloques de la I-List comienzan en el Block 2 y finalizan en la locación que se especifica (indirectamente) cuando se crea el File System.
- Varios i-nodos son almacenados en cada bloque de la I-List.

### ESTRUCTURA FÍSICA DE LOS DATA BLOCKS

- El i-nodo de cada archivo tiene la dirección apropiada del block de los bloques de datos en el File System.
- Esto no es una estructura u organización de la sección de bloques de datos en el File System.
- Los bloques de datos son los datos definidos por el usuario ya sea previamente o actualmente.
- Los bloques de datos es la sección más grande del File System.

### USO DEL SUPER BLOCK

El propósito del Super Block residente en el disco es:

- Contiene información específica del Sistema que utiliza el S.O. durante la operación de montar el File System.
- Además, contiene información específica que utiliza varios comandos de utilitarios del Sistema UNIX (Ejemplo: **fsdb**, **fsck**, etc.)

No se montan todos los campos del Super Block residente en el disco:

- CAMPO **s\_isize**: Es el primer block después del block de I-List (indicado como M) y contiene el tamaño de la I-List.
- CAMPO **s\_fsize**: Es un contador que contiene el número total de blocks en el File System (Block N).
- CAMPO **s\_tfree**: Es un contador que contiene el total de bloques libres (no utilizados) para datos en el File System
- CAMPO **s\_tinode**: Es un contador que contiene el total de i-nodos libres (no utilizados) .
- Estos dos últimos campos suelen contener información solamente para File System no montados, para File System montados los datos no son válidos porque se modifican.

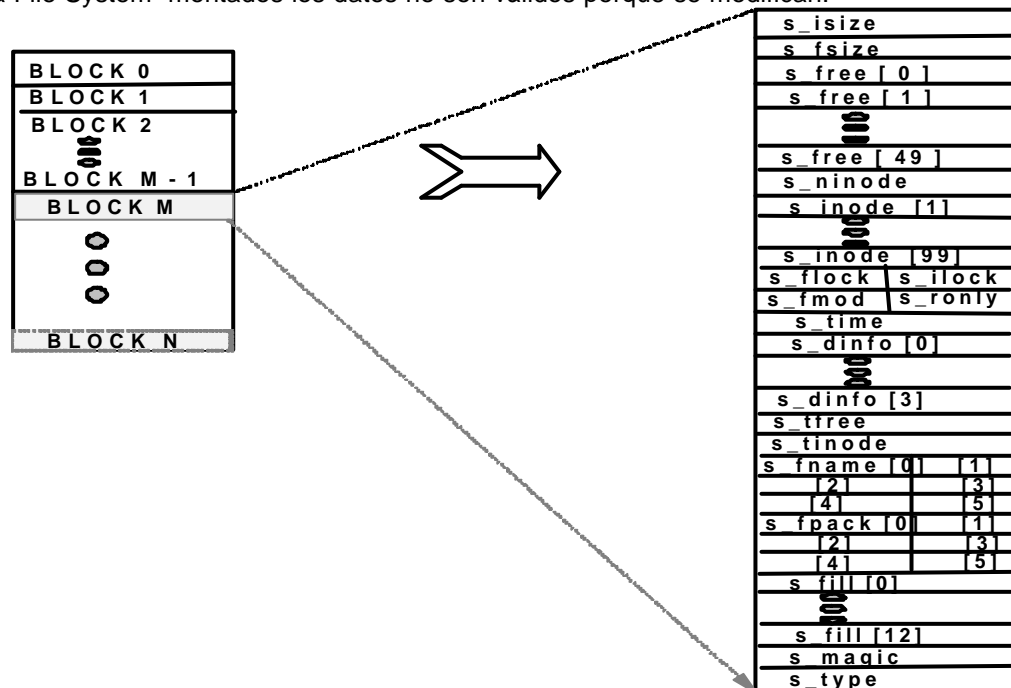


Fig 7.B.15 EL SUPER BLOCK EN EL FILE SYSTEM DE UNIX - STRUCTURE

### **CAMPO DE INFORMACIÓN:**

- **s\_fname ARRAY**: El nombre del File System (hasta 6 caracteres máximo) es almacenado en este campo.
- **s\_fpack ARRAY**: El nombre del Volumen Físico (hasta 6 caracteres máximo) es almacenado en este campo (por ejemplo: el nombre o rotulo externo del disco).
- **s\_dinfo ARRAY**: contiene información específica almacenada para el tratamiento de los dispositivos:

- **s\_dinfo [0]** = rotational gap.
- **s\_dinfo [1]** = Block/Cylinder

#### CAMPO DE TIPO DE FILE SYSTEM:

1. **s\_magic:** Si el s\_magic es FsMAGIC (Oxford187e20), entonces el File System es de estructura reciente (por ejemplo System V) y el tipo de File System va a ser indicado por el campo s\_type. El s\_type va a contener el valor de 1 para 512 Bytes por bloque y un 2 para bloques de 1024 By.
2. Si el s\_magic no es FsMAGIC, entonces el File System es asumido como para construir bloques de 512 Bytes de largo.

En realidad si vale 1: 512 Bytes, 2 : 1024 lógica y físicamente y si vale 3: 1024 lógicamente y 512 físicamente.

- **CAMPO s\_fill:** Es un array que se prefija aparte para motivar que el Super Block tenga una longitud de 512 Byte. En el futuro se piensa redefinir este campo.
- **I-List Blocks:** El número de i-nodos por I-List Blocks dependen del tamaño del bloque y de la estructura del i-nodo:
  - Un i-nodo residente en el disco generalmente ocupa 64 Bytes.
  - Un bloque del disco, o es de 512 o de 1024 Bytes.
  - $(\text{Cant.de Inodos/Block}) = (\text{Blocksize/ Size of I-node})$  (Ej:  $512/64 = 8$  o sea que caben 8 i-nodo por bloque)

#### Inodos especiales:

1. Inode 1: Corrientemente no es utilizado. Se reserva para futuras necesidades.
2. Inode 2: Es siempre el root inode del File System. Aunque este File System pueda ser montado en algún directorio externo al mismo, el inode 2 siempre será el root del File System.

#### RESIDENCIA DEL FILE SYSTEM

- Un File System puede ser creado en un dispositivo de memoria como ser un disco.
- Dispositivos de Cinta pueden ser utilizados. (El comando Volcopy es utilizado para transferencias de grandes distancias de File System o Backups de File System).
- También es posible disponer un File System residente montado en una unidad de Cinta, pero esto no es razonable por las restricciones de Entrada/Salida secuencial de los dispositivos de Cinta.
- Memoria Física (Memoria Central) puede ser utilizada para almacenar los File Systems (ejemplo el File System "tmp" en memoria central es muy común.

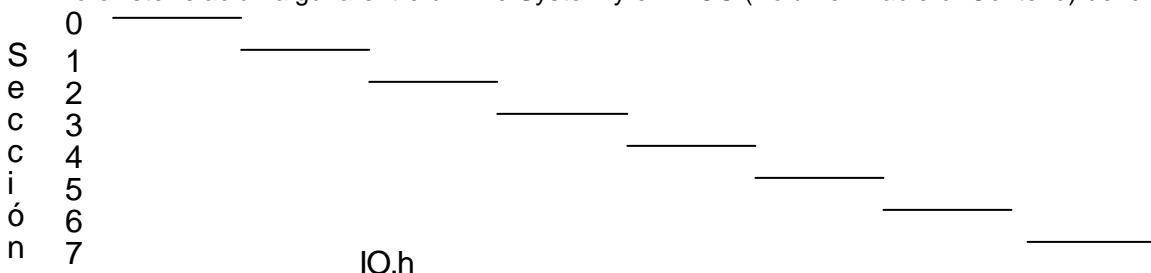
#### CREACIÓN DEL FILE SYSTEM

- El comando mkfs es utilizado por el administrador del sistema UNIX para crear el File System.

/Etc/mkfs device-name #blocks(:#inodes)

#### RELACIÓN CON EL DISPOSITIVO DISCO:

- Cada paquete de disco (disk pack) es dividido en secciones verticales cilíndricas llamadas cilindros (cylinder).
- El Administrador del sistema puede definir un grupo de cilindros como una sección.
- Existen X cilindros por dispositivo de disco.
- Pueden existir Y cilindros por sección
- Una sección puede estar parcial o totalmente asociada con un nombre de dispositivo (ejemplo: /dev/dsk0)
- Solamente un archivo puede residir en cada dispositivo.
- No existe relación alguna entre un File System y el VTOC (Volumen Table of Content) del disco.



IO.h

Fig. 7.B.16 arreglo del disco en secciones

#### SUPER BLOCK EN MEMORIA (Super Block in core)

Objetivos de este bloque de módulo:

Al completar esta parte, usted deberá ser capaz de:

1. Explicar la razón de disponer el Super Block en Memoria Central.
  2. Explicar que acciones ocurren durante la operación de montar el File System.
  3. Describir la lista de bloques libres del Super Block y su administración.
  4. Describir la lista de inodos libres del Super Block y su administración.
  5. Describir otros campos de información del Super Block.
- El Super Block de un File System, previamente montado, contiene toda la información pertinente sobre el File System.
  - Un File System es accesible únicamente después de haber sido montado.
  - Disponer el Super Block en Memoria Central en lugar de la Memoria secundaria (como ser un disco) provee al Sistema Operativo un rápido acceso a la información específica del File System, por ejemplo para actualización o accesos.
  - Ocupa un espacio en el Kernel.

#### OBJETIVO DEL MONTAJE:

- Provee acceso al File System. Un File System no puede ser accedido por el usuario si no está montado.
- La configuración de un File System jerárquico en UNIX® es fácilmente cambiabile.

#### LAS ACCIONES DE MONTAR

Antes que un File System es montado completamente, ocurren las siguientes actividades:

1. Asume que el pedido de montaje es factible.
  2. Conecta lógicamente al File System dentro de la jerarquía.
  3. Lee y lo coloca al Super Block en el espacio de direccionamiento del Kernel.
  4. Realiza la asignación de ciertos miembros del Super Block (algunos campos).
  5. Realiza la entrada en la tabla de montaje (mount table).
- Antes del montaje, el File System es física y lógicamente una jerarquía de File System.
  - Para montar se usa comandos como ser: `/etc/mount /dev/dsk0/test`

#### ESTRUCTURA DE ARCHIVOS PREVIO AL MOUNT

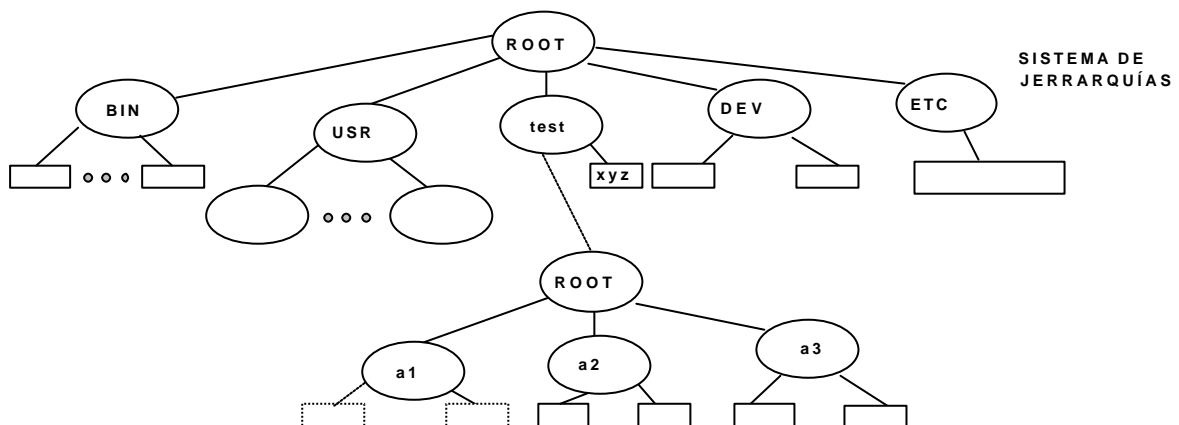


Fig. 7.B.17 File System a ser Montado.

#### El File System después que se lo monta

- Después que el montaje se completa, el File System lógico obtiene parte del Sistema jerárquico grande. A pesar de que físicamente están separados ya que el aspecto físico reside en el dispositivo de almacenamiento secundario.

#### ESTRUCTURA DE ARCHIVOS POSTERIOR AL MOUNT

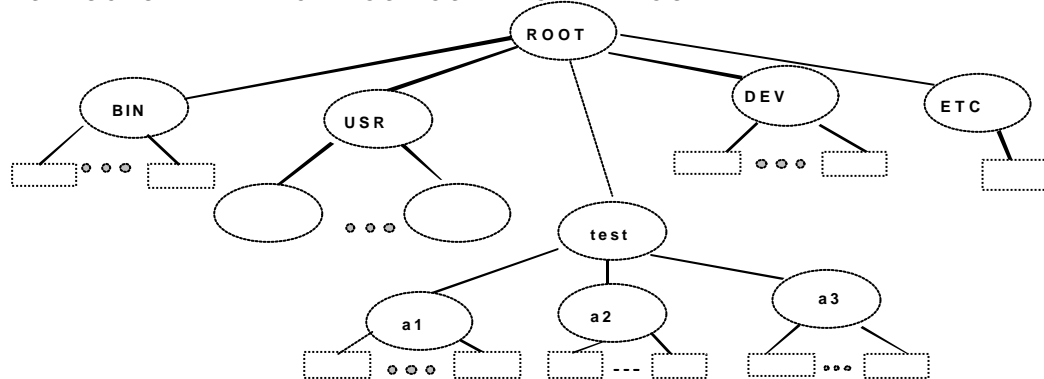


Fig. 7.B.18 File System ya Montado.

Notar la pérdida de acceso al archivo /test/xyz

### ASIGNACIÓN DEL SUPER BLOCK

- Algún campo del Super Block no es considerado significativo y/o válido cuando se trata la copia del Super Block del dispositivo - residente.
- En el instante del montaje, estos campos adquieren significado y (cuando fuera necesario) se le asigna valores válidos. Estos campos se utilizan para:
  1. Mantenimiento de la lista de bloques libres.
  2. Mantenimiento de la lista de i-nodos libres.
  3. Información sobre el almacenamiento.

### DESCRIPCIÓN DEL BLOQUE LIBRE (Free Block).

El Bloque libre:

- No es referenciado por un i-nodo.
- La disponibilidad de Bloques libres por File System para la expansión del mismo.
- Todos los bloques aparecen como una colección encadenada.

### EL ENCADENAMIENTO DE BLOQUES LIBRES:

- Es inicializado cuando se crea el File System y cuando se verifica.
- Contiene todos los bloques libres del File System.
- Consiste en una lista encadenada (Linked list) del Free Block que contiene direcciones de bloques.
- Cada direcciones de bloques contiene una dirección de 4 Byte hasta un máximo de 50 Bloques libres, más un contador que indica el número de bloques que contiene.
- La dirección cero es la dirección del próximo bloque libre que el contiene el encadenamiento.

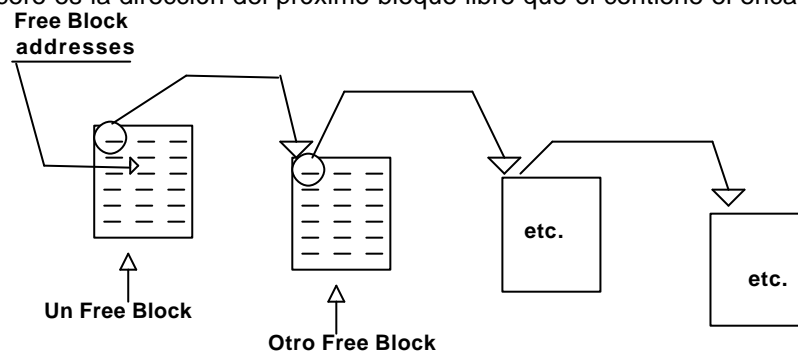


Fig. 7.B.19 Encadenamiento de Bloques libres.

### LA LISTA DE BLOQUES LIBRES EN EL SUPER BLOCK

Reside en el array **s\_free** del Super Block.

Es la cabecera del encadenamiento de los bloques libres.

Contiene un máximo de cincuenta direcciones de Bloques libres de 4 Byte cada uno.

El número de bloques libres en la lista de bloques libres es **s\_nfree**.

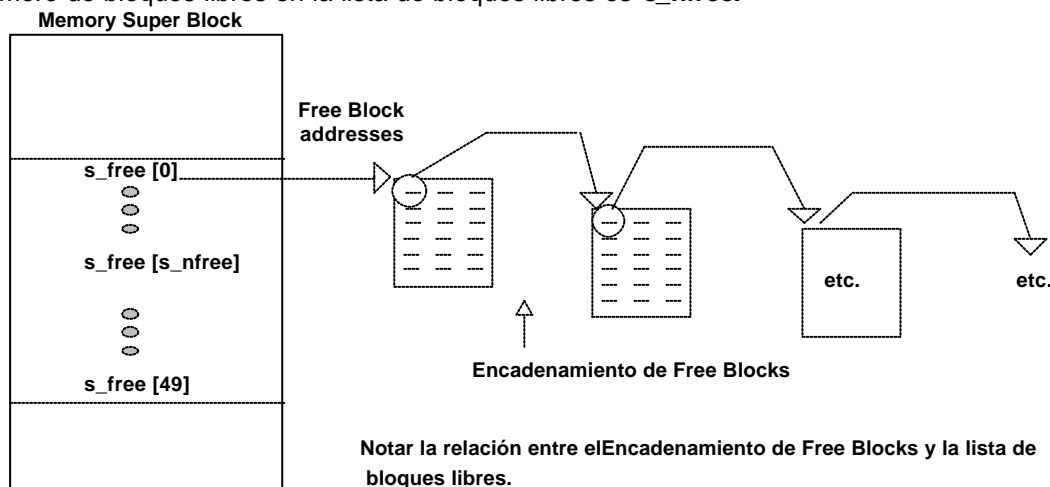


Fig. 7.B.20 La lista de Bloques libres y el free block address.

### MANTENIMIENTO DE LA LISTA DE BLOQUES LIBRES

Las razones para mantener esta lista son dos:

1. Proveer los medios para asignar los bloques a los archivos.
2. Proveer los medios para desasignar los bloques de los archivos cuando se liberan.

#### MECANISMO DE SEGURIDAD PARA LA LISTA DE BLOQUES LIBRES (LOCKING):

- La razón para Lockear es que múltiples actualizaciones simultáneas sobre esta lista es dañino y peligroso por ser datos críticos.
- **s\_flock**: cuando se comienza a actualizar la lista de bloques completa, el campo **s\_flock** es colocado como nonzero. Esto bloquea a otros procesos que intenten actualizaciones simultáneas o concurrentes.

#### MECANISMOS DE ASIGNACIÓN

```
alloc(dev)
{
  IF dev IS MOUNTED
    IF FREE LIST NOT IN USE
      IF s_nfree>1          /* cuando hay más de un bloque libre en la lista*/
        DECREMENT s_tfree  /* decrementa s_nfree*/
        RETURN s_free [--s_nfree] /* y retorna s_free [--s_nfree] bloque */
      else
        FREE LIST UNDERFLOW
    }
}
```

#### FREE BLOCK LIST UNDERFLOW

- Cuando hay solamente un block en la lista de bloques libres es el “address-holding” free block.
- Usa el contenido de el “address-holding” free block. para refrescar (actualizar) la free list.
- Retorna al “address-holding” block.

```
(FROM alloc)    GIVEN THAT s_nfree = 1
  s_flock ++;
  COPY s_free[0] CONTENTS (ADDRESS) INTO FREE LIST/*lee entrada s_free[0] */
  COPY s_free[0] “nsfree” VALUE TO s_free
  s_flock =0;
  RETURN “old” s_free[0]
```

#### MECANISMO DE DESASIGNACIÓN de BLOCK

- Cuando la Lista de Bloques Libres no esta llena:
- Adiciona un bloque a la lista de libres.
- Incrementa **s\_nfree**
- Incrementa **s\_tfree**.

```
free (dev, block number)
  IF FILE SYSTEM IS MOUNTED
    IF FREE LIST IS NOT LOCKED
      IF s_nfree<50
        s_tfree++
        s_free[s_nfree++] = BLOCK NUMBER
      else
        FREE LIST OVERFLOW
```

#### FREE BLOCK LIST OVERFLOW

- Cuando la Lista libre está llena hace:
  - Lockea a la lista.
  - Crea un address-holding block.
  - Coloca el block en s\_free[0].
  - Resetea el contador de free block.
  - Deslockea a la lista.
- ```
(FROM free)    GIVEN s_free = 50
  s_flock++;
  COPY FREE LIST CONTENTS TO FREE BLOCK;
  s_nfree=0;
  s_free[ s_nfree++] = FREE BLOCK;
  s_tfree++;
```

```
s_flock = 0;
```

### LISTA DE INODOS LIBRES

- Una lista de i-nodos libres es un i-nodo con un modo de cero.
- Un inodo libre es alocado cuando se crea un archivo.
- Tiene un máximo de un número de 100 i-nodos no alocados.
- La lista de i-nodos libres es el arreglo **s\_inode** en el super bloque.
- El número de i-nodos libres en la lista de i-nodos libres es el campo **s\_ninode**.

### MECANISMO DE SEGURIDAD PARA LA LISTA DE I-NODOS LIBRES (LOCKING):

- La razón para Lockear es que múltiples actualizaciones concurrentes sobre esta lista es dañino y peligroso por ser datos críticos.
- **s\_iloc**: cuando se comienza a actualizar la lista de bloques completa, el campo **s\_iloc** es colocado como nonzero. Esto bloquea a otros procesos que intenten actualizaciones simultáneas.
- Este flag es usado con **sleep()** y **wakeup()**.

### MECANISMOS DE ASIGNACIÓN de INODO

- Cuando la lista de inodos libres no está llena:

```
ialloc(dev, mode, nlink)
{
  IF FILE SYSTEM IS MOUNTED
    IF INODE LIST IS NOT LOCKED
      s_iloc++;
      IF s_ninode>0
        READ IN s_inode[s_ninode-] /* Lee el Próximo inodo lleno de la lista*/
        INITIALIZE INODE           /*inicializa el inodo*/
        s_iloc=0;
        RETURN INODE POINTER      /*Retorna el Puntero*/
      else
        /* INODE LIST UNDERFLOW*/
    }
}
```

### INODE LIST UNDERFLOW

Cuando la lista de i-nodos está llena:

- Busca en el residente del disco el contenido del I-list blocks de los 100 i-nodos libres.
- Lo lee en el campo inodo s\_inode [99]
- Inicializa el i-nodo y retorna el puntero.

(FROM ialloc) GIVEN s\_ninode = 0

STARTING WITH s\_inode [0] /\* I-Nodo más pequeño no es retornado\*/

```
s_flock ++;
READ I-LIST BLOCKS
FILL INODE LIST WITH FREE INODE NUMBER UNTIL 100 FREE INODES
FOUND;
s_ninode = NUMBER OF INODES FOUND;
READ s_inode[s_ninode-]
s_flock =0;
RETURN inode POINTER;
```

### MECANISMO DE DESASIGNACIÓN de INODO

Cuando la Lista de i-nodos no esta llena:

- Adiciona el i-nodo libre a la lista. Si la Lista de i-nodos esta llena, coloca al i-nodo liberado como elemento cero si menor que el i-nodo corriente cero. De lo contrario no hace nada con el i-nodo liberado.

ifree (dev,inode number)

```
IF FILE SYSTEM IS MOUNTED
  IF INODE LIST IS NOT LOCKED
    IF (s_ninode >= 100)
      { IF (s_inode [0]> INODE NUMBER)
        { s_inode[0] = INODE NUMBER
          RETURN
        }
      }
    }
```

```

    }
else
    { s_inode[ s_ninode++] }

```

### OTROS CAMPOS DE INFORMACIÓN

- **s\_fmod:** Cuando el superblock es modificado, este campo es seteado a uno. Ni bien se carga el File System, se escribe en **sync()**. La próxima llamada de **sync()** va a causar que el super block será escrito en el disco.
- **s\_roonly:** Si el File System será montado como solamente de lectura, entonces este campo será seteado en 1.
- **s\_time:** La última vez que el superblock es escrito en el disco se setea **s\_time** con la fecha y hora que ocurrió la grabación.

### LA TABLA DE INODOS

- Los propósitos de la Tabla de inodos son dos:
  1. Contiene copia de los inodos activos residentes en el disco (campo **i\_list**) de todos los File System montados, para rápida referencia o modificación.
  2. Provee un punto de control simple para los diferentes procesos.
- La Tabla de inodos es accesible solamente vía System Call.
- Las acciones que requiere una entrada en la Tabla son:
  - ♦ **Semipermanente:**
    - ◇ I/O (**open, read, write, close**).
    - ◇ Creación de Archivos.
    - ◇ Cambio del Directorio Actual.
    - ◇ Montaje del File System.
    - ◇ Ejecución del Archivo.
  - ♦ **Temporarios:**
    - ◇ Obtener el estado del Archivo.
    - ◇ Cambios del estado del Archivo.
    - ◇ Vinculación/Desvinculación de Archivos.
- El root inode es la única entrada permanente en la Tabla.

### LA ESTRUCTURA DE LA TABLA DE INODOS

- La Tabla de inodos es un arreglo de la estructura de inodos que reside en la Memoria Central en el espacio de direcciones del kernel.
- El nombre del arreglo es **inode [ ]**.

### TAMAÑO DE LA TABLA DE INODOS:

- El número de entradas en la Tabla de inodos (usualmente entre 100 y 300) es definido por el administrador del Sistema en el momento de la generación del sistema.
- Este número corresponde al total de inodos que requieren acceso simultáneo.

### RELACIÓN CON LA I-LIST:

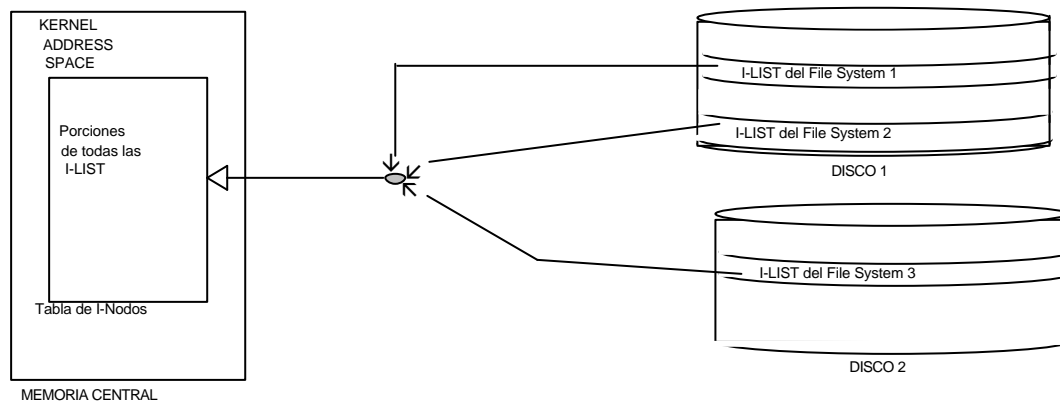


Fig. 7.B.21 Relación entre La I-List y la tabla de inodos.

Notar que la Tabla de inodos del Sistema contiene copias de inodos de varias I-LIST.

## COMPARACION DE INODOS RESIDENTES EN MEMORIA Y EN DISCO

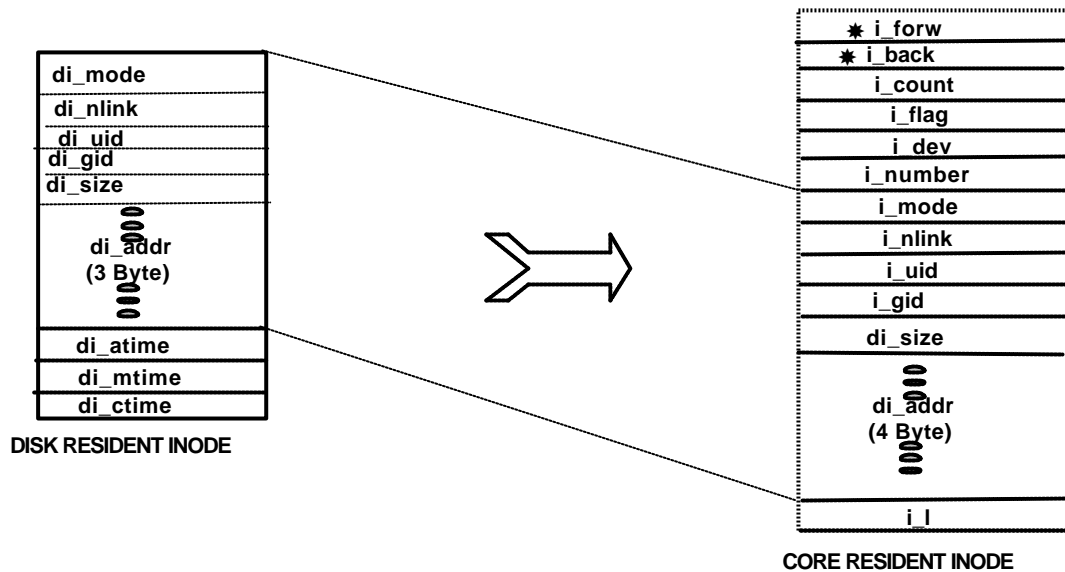


Fig. 7.B.22. comparación de inodos residentes en memoria y en disco

## PUNTEROS DE ENCADENAMIENTO DE INODOS:

- Un juego de punteros utilizados o para hash list pointers o para puntero a la lista disponible.
- Cada estructura de inodos es siempre incluido en un encadenamiento doblemente vinculado que son:
  - ♦ encadenamiento de hash.
  - ♦ encadenamiento de inodo disponible.
  - ♦ estos punteros de encadenamiento de inodos apuntan a la estructura de inodo próximo y al previo incluido en el respectivo encadenamiento.

## CAMPO CONTADOR DE REFERENCIAS

- **i\_count** refleja el número total de accesos actuales para usar el inodo

## CAMPO BANDERAS (FLAG)

- Refleja el estado de la estructura de inodos y se registran en **i\_flag**.

- Las banderas son:

**ILOCK** - si el inodo está o no lockeado.

**IUPD** - El archivo fue modificado.

**IACC** - El tiempo de acceso fue actualizado

**IMOUNT** - El inodo fue montado encima.

**IWANT** - El proceso está dormido esperando por un inodo.

**ITEXT** - Prototipo de inodo de puro texto.

**ICHG** - El inodo fue modificado.

**ISYN** - Realiza una escritura sincrónica.

## IDENTIFICACIÓN DEL INODO:

- Se puede identificar a cada inodo en la tabla de inodos usando los campos **i\_dev** y el **i\_number**.
- El campo **i\_dev** es el mayor y menor número de dispositivo que identifica el dispositivo (file System) en el cual el inodo reside.
- El campo **i\_number** es el número de inodo respectivo que ha comenzado en la I-LIST.

## CAMPOS COMUNES DE LA ESTRUCTURA DE INODOS.

- Los siguientes campos de la estructura de inodo residente en Memoria Central son semejantes a los campos de inodos residente en el disco:

**i\_mode** (**di\_mode**)

**i\_nlink** (**di\_nlink**)

**i\_uid** (**di\_uid**)

**i\_gid** (**di\_gid**)

**i\_size** (**di\_size**)



**PUNTERO DE BLOQUES:**

- El campo **i\_blks** del inodo residente en memoria central contiene información sobre la dirección del bloque en dos diferentes formatos.
- El formato depende del tipo de archivo representa el inodo: regular, directorio, especial o FIFO

**TAMAÑO DEL PUNTERO DE BLOQUES:**

- El puntero de bloques residente en memoria central es asignado como long words (4Bytes).
- El puntero de bloques residente en disco es asignado como 3 caracteres (3 Bytes).
- Cuando el puntero de bloques es leído del inodo residente en disco y cargado en memoria central, el kernel convierte la dirección de 3 Bytes a una dirección de 4 Bytes asignando un valor cero al byte de orden superior en el inodo residente en memoria central.

**PUNTERO DE BLOQUES DE ARCHIVOS REGULARES / DIRECTORIOS**

- Las direcciones de bloques almacenados en archivos regulares y directorios están contenidos en el arreglo **i\_blks.ip.i\_a** (**i\_addr** es definido como **i\_blks.ip.i\_a**).
- Este arreglo contiene 13 elementos que corresponden exactamente a los punteros de bloques directos, indirectos, doble indirecto y triple indirecto como se muestra en las Figuras 7.57, 7.62, 7.63 y 7.64.

**PUNTERO DE BLOQUES DE ARCHIVOS ESPECIALES:**

- Los i-nodos de archivos especiales (device) no contienen punteros de bloques. Solamente los 2 Bytes de orden inferior del arreglo **i\_addr [0]** contienen la información de identificación del dispositivo.
- Byte 0 contiene el número de dispositivo mayor.
- Byte 1 contiene el número de dispositivo menor
- Byte 2 y 3 contienen valores = 0.

**PUNTERO DE BLOQUES DE ARCHIVOS FIFO:**

- Solamente los 10 direcciones (directo) de bloques pueden ser almacenadas en el i-nodo.
- Los otros campos de direcciones de bloques contienen la siguiente información:
  1. Read pointer (**i\_frptr**)
  2. Write pointer (**i\_fwptr**)
  3. Count of reading process (**i\_frnt**)
  4. Count of writing process (**i\_fwcnt**)
- Information flags (**i\_fflag**)

| i-node            |                |
|-------------------|----------------|
| ○<br>○<br>○       |                |
| <b>i_addr [0]</b> |                |
| <b>i_addr [1]</b> |                |
| <b>i_addr [2]</b> |                |
| <b>i_addr [3]</b> |                |
| <b>i_addr [4]</b> |                |
| <b>i_addr [5]</b> |                |
| <b>i_addr [6]</b> |                |
| <b>i_addr [7]</b> |                |
| <b>i_addr [8]</b> |                |
| <b>i_addr [9]</b> |                |
| No Usado          | <b>i_frptr</b> |
| <b>i_fwptr</b>    | <b>i_frnt</b>  |
| <b>i_fwcnt</b>    | <b>i_fflag</b> |
| <b>i_l</b>        |                |

Fig. 7.B.23 Punteros de bloques de archivos FIFO

**PUNTEROS DE BLOQUES DE ARCHIVOS FIFO**

- Notar que el campo de información es pequeño en cambio el de direcciones es grande.

**ÚLTIMO BLOQUE LÓGICO LEÍDO:**

- Para acelerar la Entrada/Salida de un Archivo, el Sistema determina si puede leer el próximo trabajo al mismo tiempo que establece la lectura del presente trabajo.
- **i\_blks.i\_l** contiene el último bloque leído exitosamente para el presente i-nodo.
- **i\_lastr** es definido como **i\_blks.i\_l**

**LISTA DE VINCULACIÓN DE LA TABLA DE INODOS:**

- Dos listas doblemente vinculadas son utilizadas por el sistema para mantener las entradas de la Tabla de i-nodos. Se usan solamente un juego de punteros para una u otra lista.
- Estas listas son:
  1. Lista vinculada disponible.
  2. Lista vinculada de Tabla Hash.

**LISTA DE VINCULACIÓN DE ENTRADAS DE LA TABLA DE INODOS:**

- Contiene todas las ranuras (slots) no utilizadas en la Tabla de i-nodos. Cuando el contador de referencia de un i-nodo en la tabla de i-nodos llega a cero, la ranura es vinculada a la lista de vinculación de entradas de la tabla de i-nodos vía los punteros **i\_avforw** y **i\_avback** en el i-nodo.
- La lista se mantiene en modalidad LIFO (Last In First Out).
- El comienzo de las entradas de la lista vinculada disponible es la variable ifreelist, que actualmente es el puntero al último i-nodo adicionado a la lista.
- La inicialización de todas las entradas a la lista vinculada disponible toma lugar en el instante de "booteo".

**TABLA DE HASH DE INODOS.**

- Es usado para almacenar inodos. Cuando una ranura de la tabla de inodos es alocado, se remueve de la lista de disponibles y es colocado en la lista de hash y es encabezado con una entrada en la tabla de hash (**hinode [ ]**)
- Encabezado de la lista vinculada: La entrada a la tabla de hash es usada para buscar la lista vinculada de hash apropiada para hallar un i-nodo.
- El tamaño de la lista de hash es NHINO, usualmente 128 (debe ser una potencia de 2).

**MECANISMO DE LOCALIZACIÓN DE INODOS.**

- Utilizando el número de i-nodo, el algoritmo de hashing produce el índice en la tabla de hash para comenzar la lista de hash apropiada.

```
{
iget (dev, ino)
  DETERMINE HASH TABLE ENTRY
  WHILE NOT THE END OF HASH LIST
  COMPARE dev AND ino INFORMATION.
  IF INODE WAS FOUND IN LIST
  RETURN (INODE PTR)
  ELSE
    /* inode allocation*/
}
```

- Si el i-nodo esta en la lista apropiada, el puntero a la entrada de la lista de i-nodos es retornado
- Si el i-nodo no esta en la lista, no se busca en otra lista de hash. Como consecuencia se crea una entrada nueva en la presente lista de hash.

**MECANISMO DE ALMACENAMIENTO (ALLOCATION) DE INODOS:**

- Selecciona la entrada a la tabla de i-nodo apuntado por ifreelist. Esta es la última entrada libre.
- Remueve el i-nodo de la lista de disponibles y lo modifica de acuerdo a lo explicado.
- Vincula el puntero del i-nodo al final de la lista de hash.

**LA TABLA DE ARCHIVO (FILE TABLE).**

- El propósito de la tabla de archivos es contener información específica de cada archivo abierto.
- Las entradas son realizadas como resultado de la ejecución de los siguientes llamadas al sistema (system calls):
- **open ()**      · **create ()**      · **pipe ()**.

**LA ESTRUCTURA DE LA TABLA DE ARCHIVO (FILE TABLE STRUCTURE).**

- Está localizada en el kernel: La tabla de archivos es un arreglo de una estructura de tabla de archivos que reside en el espacio de direccionamiento del kernel.
- El nombre del arreglo es **file** [ ].

**TAMAÑO DE LA TABLA DE ARCHIVO:**

- El número de entradas en la Tabla de Archivo (usualmente entre 100 y 300) es definido por el administrador del Sistema en el momento de la generación del sistema.
- Este número corresponde al total de archivos abiertos en todo el sistema.

**LAS ENTRADAS A LA TABLA DE ARCHIVO:****CAMPO FLAG:**

- Cada entrada a la Tabla de archivo tiene una bandera de estado (**f\_flag**). Esta bandera indica:
- El estado de abierto para ésta entrada (Por ejemplo: abierto para lectura, escritura, etc.).
- Es seteado cuando se invocan los siguientes System Calls: **open**, **create**, **pipe**, **close** y **fcntl**.

**REFERENCE COUNT:**

- Tiene como propósito reflejar el número total de descriptores de archivos que han sido alocados por los procesos (o el proceso) al acceder al archivo vía esta entrada de la tabla de archivo.
- El contador de referencias esta contenido en **f\_count** y es alterado por las invocaciones de los system calls **open**, **create**, **dup**, **fcntl**, **close** y **pipe**
- **open**, **create** y **pipe**  $\triangleright$  **f\_count** = 1; **dup**, **fcntl** y **fork**  $\triangleright$  **f\_count** ++; **close**  $\triangleright$  **f\_count** --.

**PUNTEROS:**

- **f\_up** contiene valores de puntero. El puntero es accedido por:
- ♦ **f\_up.f\_uinode** (definido como **f\_inode**) cuando el contador de referencia es no cero.
- ♦ **f\_up.f\_unext** (definido como **f\_next**) cuando el contador de referencia es cero.
- ♦ **f\_inode** apunta al correspondiente inodo.
- ♦ **f\_next** apunta a la próxima entrada disponible de la tabla de archivo.

**FILE OFFSET:**

- El propósito de **f\_un.f\_off** es reflejar la posición corriente dentro del archivo que fuera abierto para una Entrada / Salida.
- Este número es compartido por todos los procesos que han accedido al inodo a través de la entrada de la tabla de archivo.
- Las acciones que modifican el desplazamiento son realizados por los system calls **read**, **write**, y **lseek**.
- **f\_un.f\_off** es definido como **f\_offset**.

**RELACIONES ENTRE TABLAS:****DESCRIPCIÓN DE LAS RELACIONES GENERALES**

- Una entrada en bloque del usuario (user block) apunta a cada entrada de la tabla de archivos alocados.
- Cada entrada de la tabla de archivos alocados apunta a una entrada en la tabla de inodos.
- Una entrada de la tabla de archivos no alocados apunta a otra entrada de tabla de archivo que tiene una vinculación en la lista de libres (ffreelist).

**RELACIÓN CON EL BLOQUE DEL USUARIO (USER BLOCK)**

- Cada invocación de **open**, **create**, **dup**, **fcntl**, y **pipe** retornan uno (o dos) descriptores de archivo, que son índices en el arreglo **u.u\_ofile**.
- Cada elemento en el arreglo **u.u\_ofile** contiene un puntero a la correspondiente entrada de la tabla de archivo para su apertura.

**RELACIÓN CON LAS TABLAS DE INODOS**

1- **open()** singular por el proceso 2.

2- **open** seguido por un **dup()** por el proceso 1.

3- **open()** por el proceso 2 seguido por un **fork ()** para crear el proceso 3.

4- Ejemplo de un inodo que fuera abierto más de una vez (en este caso por más de un proceso).

- Cada entrada de la tabla de archivos tiene un elemento (**f\_up.f\_uinode**) que contiene el puntero a la correspondiente entrada en la tabla de inodo.
- El elemento **f\_up.f\_uinode** es comúnmente definido como **f\_inode**.

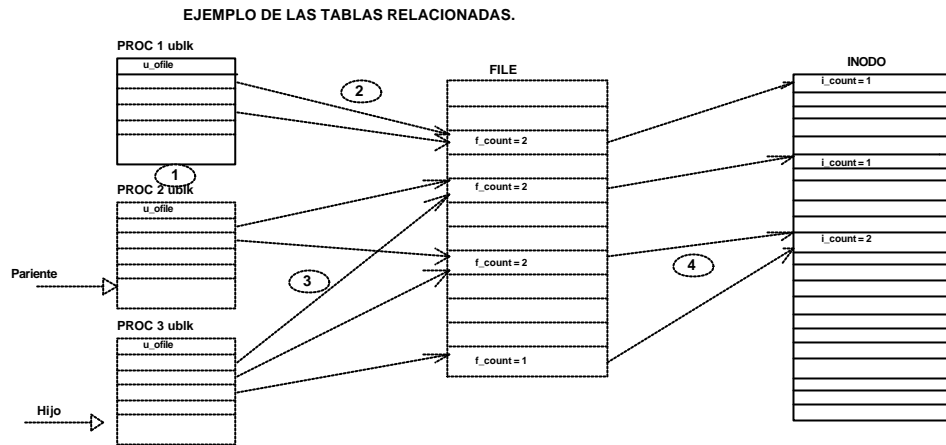


Fig. 7.B.24 Ejemplo de tablas de procesos , de archivo y de inodos relacionados

**ADMINISTRACIÓN DE LA TABLA DE ARCHIVO:**

- Para alocar (asignar) la entrada de la tabla de archivo, el valor en el campo **ffreelist.f\_next** es retornado y la lista de libres (**freelist**) es modificada para reflejar el cambio.
- Para desalocar (desasignar) la entrada de la tabla de archivo, la entrada es colocada en la Lista de libres (**freelist**) así el puntero **ffreelist.f\_next** apunta a si mismo y también apunta al previo **f\_next**

**MONTANDO EL SISTEMA DE ARCHIVOS:**

- Un Sistema de archivo (que fuera previamente creado o existe) reside en un dispositivo lógico separado (y posiblemente físico también).
- Un sistema de archivos es montado para proveer información a las subrutinas internas que permitirán a los usuarios acceder vía el camino del sistema de archivos de UNIX (UNIX File System pathname) sin consideración en cual dispositivo reside el archivo.

**PROPOSITO DE LAS TABLAS DE MONTAJE (MOUNT TABLES):**

- Registra la presencia de un dispositivo montado: Cada entrada activa en la tabla de montaje representa un sistema de archivo montado.
- La entrada de la tabla de montaje tiene dos principales usos:
- Palabra para la resolución del camino (pathname resolution): Para resolver el camino cuando la estructura requiere un cambio del dispositivo residente (por ejemplo nombre lógico se ha extendido más allá de un sistema de archivo)
- Contiene un puntero al Super Block: Permite localizar el super block para el uso de las subrutinas internas.

**TAMAÑO Y ESTRUCTURA:**

- La tabla de montaje es un arreglo de una entrada a la estructura de la tabla de montaje, ninguna de las cuales es vinculada.
- Tamaño: El número de entradas de la tabla de montaje resulta igual al número de file system montados simultáneamente.
- Este valor está usualmente comprendido entre 12 y 18 y es definido por el administrador del Sistema en el momento de la generación del sistema.

- **MOUNT TABLE ENTRY**

|                |
|----------------|
| <b>m_flags</b> |
| <b>m_dev</b>   |
| <b>m_inodp</b> |
| <b>m_bufp</b>  |
| <b>m_mount</b> |

Fig. 7.B.25 Entrada a la tabla de montaje

**CAMPO: m\_flag**

- Refleja el estado de la entrada como:

- en uso
- disponible
- intermediado ( durante el montaje y el desmontaje)

**CAMPO: m\_dev:**

- Identificación del dispositivo: el mayor y menor número de dispositivo es almacenado en la entrada **m\_dev**.
- Es usado para la localización del Super Block: La subrutina **getfs ()** es usada por cualquier subrutina que accede al super block. La subrutina **getfs ()** usa el campo **m\_dev** para encontrar la entrada apropiada de la tabla de montaje.
- Es usado para resolver caminos (pathnames): cuando se realiza un file pathname hacia una dirección de referencia (por ejemplo hacia /), el campo **m\_dev** es usado para eventualmente redireccionar la búsqueda a otro sistema de archivo.

**CAMPO: m\_inodp:**

- Puntero al directorio "montado": El campo **m\_inodp** contiene la dirección de la entrada en la tabla de inodo que tiene "sobre montado" el inodo del directorio.
- Cuando es usado un camino para localizar un archivo ( desde el root hacia abajo), si un inodo es encontrado que tiene el bit de montaje colocado en in y su campo flag, la búsqueda es redirigida a través de la tabla de montaje para localizar cual entrada tiene un campo **m\_inodp** apuntando al inodo.

**CAMPO: m\_mount:**

- Apunta al inodo raíz (root inode) del Sistema de archivo montado: El campo **m\_mount** contiene la dirección de la entrada en la tabla de inodo que tiene el inodo raíz del sistema de archivo montado.
- Si el camino buscado es redireccionado a través de la tabla de montaje, el va a contener el inodo que está apuntado por **m\_mount**.

**CAMPO: m\_bufp:**

- Puntero al buffer que contiene el super block: La dirección del buffer que contiene el super block del file system montado es almacenada en el campo **m\_bufp**.
- Usado por rutinas internas: El buffer que contiene el super block no es colocado en una cadena de buffers. Entonces el campo **m\_bufp** provee la única manera de acceder al super block. Estos accesos son realizados por varias subrutinas internas.

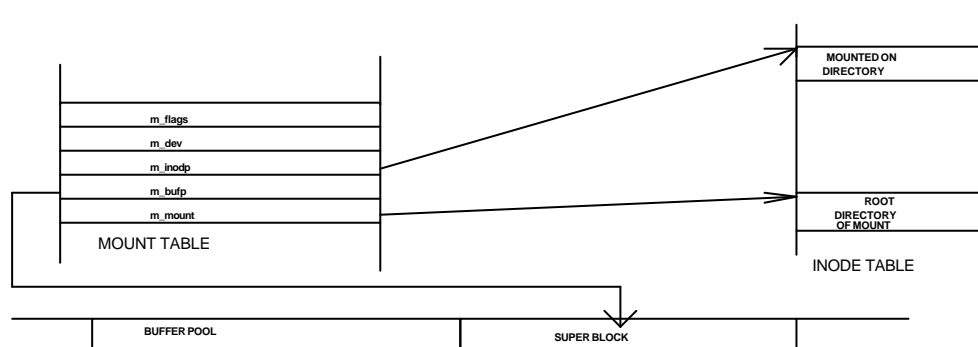
**RELACIÓN CON OTRAS TABLAS:**

Fig. 7.A.26 Relaciones en las tablas de montaje, inodo y el super block.

**Anexo 7.C. : FILE SYBSTEM EXT2****LAS VENTAJAS DE LINUX**

- Es gratis (monetaria y legalmente)
- Interoperabilidad
  - Filesystem: ext2, fat, vfat, minix, hpfs, ntfs, fat32, hfs, bsdm
  - Network Filesystems : NFS, SMB/CIFS, ncpfs, cryptographics NFS
  - Network File Serving: NFS, SMB/CIFS, Appletalk, Netware, HTTP
  - Binary Compatibility: SCO, Solaris, SunOS, Digital Unix
  - Multi-Bootting: LILO, loadlin, MILO, SILO, etc.
- Soporte para múltiples plataformas
- Networking
  - Routing
  - IP Masquerading
  - Firewalling

- Y muchas mas

## BREVE COMPARACIÓN CON OTROS SISTEMAS OPERATIVOS

MS-DOS corre solo en procesadores de la línea x86, no tiene soporte para múltiples usuarios ni tampoco es multi-tasking (multitarea), tampoco es gratis. También posee una pobre Interoperabilidad con otros sistemas operativos y no incluye software para Networks, ni programas de desarrollo, ni muchos de los utilitarios disponibles para Linux

Microsoft Windows ofrece algunas de las capacidades gráficas de Linux y posee algunas capacidades de network, pero sufre de todas las otras desventajas de MS-DOS antes mencionadas.

Windows NT esta disponible para la plataforma Digital Alpha, como también para procesadores x86, pero sufre de todas las desventajas de Windows a saber: bugs, virus, altos precios.

## HISTORIA

El primer sistema de Archivos que soportó Linux fue el MinixFS, implementado por Linus Torvalds mientras desarrollaba Linux a partir de Minix, y que tenía varias limitaciones: no podía considerar nombres de archivo de más de 14 caracteres (lo que aún así era bastante mejor que los clásicos 8 + 3), el tamaño máximo de los archivos era de 64 Mb y no tenía un rendimiento muy alto.

Cuando Torvalds portó el sistema de Archivos de Minix a Linux, añadió una nueva extensión a éste que le permitía soportar nombres de Archivos de hasta 30 caracteres, aunque esto no mejoraba mucho la situación.

Por estas razones y otras motivaciones personales, Rémy Card, desarrolló en 1992 el primer sistema de archivos nativo de Linux, el ExtFS (Extended File System), también a partir del código del MinixFS. En éste, un archivo podía tener un nombre de longitud variable de hasta 256 caracteres y un tamaño máximo de 2 Gbytes. Sin embargo, tenía un gran caballo de batalla en la administración del espacio libre: ExtFS almacenaba las referencias a bloques libres en una lista, existía una gran tendencia a la fragmentación externa del sistema de archivos y era más lento que MinixFS.

Simultáneamente, Frank Xia desarrolló e implementó XIA FS, más simple, eficiente y también basado en MinixFS. Poseía estructuras de control más grandes, entradas de directorio de longitud variable y rutinas de asignación optimizadas. Muy similar a ExtFS, su rendimiento era superior (sobre todo por sustituir el empleo de listas para la gestión de los bloques libres por el uso de bitmaps).

En 1993, Rémy Card y Wayne Davidson introdujeron una variante de ExtFS denominada Ext2FS, mejorada a su vez por otros programadores como Stephen Tweedie y Theodore Ts'o. Ext2 es un sistema de archivos nativo de Linux que proporcionan la mayoría de las distribuciones de este sistema operativo por defecto debido a sus indudables ventajas sobre otros.

## INTRODUCCIÓN

Trataremos en este apartado del sistema de archivos de uno de los sistemas operativos más aceptados en los últimos tiempos: el Ext2FS de Linux. Como todo sistema de archivos que se precie, ofrece funciones para la creación y eliminación de archivos y directorios, la lectura y escritura de datos en archivos, la protección de la información, etc.

Uno de los objetivos principales que se plantearon en su diseño fue el de conseguir la independencia del dispositivo para que, de esta forma, las operaciones de acceso a los archivos fuesen las mismas, de manera independiente de su localización efectiva (disco rígido, disquete, CD-ROM, etc.). Dando un paso más allá, Ext2FS permite que el acceso a los dispositivos de entrada y salida se realice de igual forma que el acceso a los archivos ordinarios.

Entre las características más importantes de Ext2 figuran las siguientes:

- Archivos tipo UNIX, distinguiendo entre Archivos, directorios, Archivos de dispositivo y enlaces simbólicos.
- Particiones grandes, de hasta 2 Tb.
- Nombres de Archivos de hasta 255 caracteres.
- Reserva de bloques para el administrador del sistema (*root*), de un 5% por defecto.
- Enlaces simbólicos rápidos.
- Tamaño del bloque seleccionable durante la creación del sistema de Archivos.
- Detección del estado del sistema y contabilidad del número de montajes por medio de fsck.
- Semánticas de BSD o System V R4 como opción durante el montaje.
- Actualizaciones síncronas tipo BSD como opción durante el montaje.
- Parámetros ajustables (conteo de montajes, intervalo entre chequeos y número de bloques reservados).
- Comportamiento ante errores ajustable, bien durante el montaje (*mount*), bien durante el refinamiento del sistema (*tune2fs*): continuar tras el error, remontar el sistema de Archivos en sólo lectura, modo pánico.

Así mismo, durante su diseño se planteó el llevar a cabo las siguientes extensiones:

- Recuperación de Archivos borrados.
- Listas de control de acceso.
- Compresión automática de Archivos.
- Fragmentos de bloque.

Estas nuevas extensiones debían ser aceptadas una vez el sistema hubiese entrado en servicio, de forma que se pudiese ampliar su funcionalidad aplicando un esfuerzo mínimo.

### ESTRUCTURA DEL FILE SYSTEM

En la figura siguiente se muestra cómo se relacionan los sistemas de Archivos en Linux con otros elementos del núcleo:

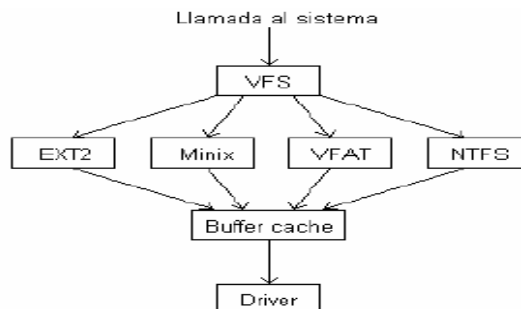


Fig. 7.C.1 Relación del File System de Linux con elementos del Kernel

#### Estructura global del Sistema de archivos de Linux-

El elemento más en contacto con los procesos de usuario es la interfase de llamadas al sistema. Por el lado opuesto, en el nivel más cercano al hardware tenemos los controladores de dispositivo o *drivers*.

- Llamadas al sistema: Todas las llamadas relacionadas con el sistema de archivos (apertura, lectura, escritura, creación, etc.) son dirigidas a la capa VFS, en lugar de a los sistemas de Archivos directamente.
- VFS (*Virtual File System*): Es la capa de abstracción que, actuando como colchón, analiza las peticiones realizadas y las pasa al sistema de archivos apropiado para que las resuelva. Esto permite que en Linux puedan convivir varios sistemas de Archivos de forma concurrente. VFS trabaja sólo con estructuras genéricas (superbloques, i-nodos y Archivos) y con punteros a funciones (operaciones con superbloques, i-nodos y Archivos).
- Buffer cache: Sirve para acelerar los accesos a disco, tanto en lectura como en escritura, almacenando los últimos bloques que han sido accedidos.
- Driver: O controlador de dispositivo, es el encargado de actuar sobre el dispositivo físico concreto donde se almacenan los Archivos (disco, disquete, CD-ROM, etc.), respondiendo a las peticiones realizadas desde las capas superiores y ocultándoles las peculiaridades hardware. De este modo, el Sistema de Archivos no sabe nada acerca de pistas, cabezas o sectores. Simplemente percibe el disco (o el dispositivo de almacenamiento secundario que se emplee como elemento de almacenamiento masivo) como una secuencia numerada de bloques de tamaño fijo.

Para el sistema, cada disco no es más que un dispositivo lógico (por ejemplo, /dev/hda1, /dev/hda2, /dev/sda1, etc.) con dos números de dispositivo asociados (major number y minor number). Estos números se utilizan como índices dentro de una tabla de funciones del núcleo, la cual permite localizar el controlador concreto que debe emplearse para manipular el disco en cuestión.

Los datos que lee y escribe el sistema de Archivos desde y sobre el dispositivo físico por medio del *driver* se toman siempre en bloques de tamaño fijo que constituyen las unidades lógicas mínimas de información. Dicho de otro modo, si solicitamos la lectura de un solo byte de un Archivo, el *driver* lee el bloque entero donde se almacena dicho byte; lo mismo sería aplicable a los procesos de escritura. El *driver* proporciona primitivas para leer o escribir uno o varios bloques en cada operación de lectura/escritura. El tamaño de estos bloques puede ser escogido (1, 2 ó 4 Kb) en el momento de la creación del sistema de Archivos.

Trabajar con bloques de tamaño fijo tiene como inconveniente la posible fragmentación de los bloques. Si, por ejemplo, el tamaño del bloque es de 1 KiBy y un Archivo ocupa 2049 bytes, sería necesario asignarle tres bloques, aunque del último sólo empleásemos 1 byte. Este problema puede reducirse haciendo que el tamaño del bloque sea más pequeño, pero ello provocaría una degradación del rendimiento del sistema de Archivos por ser necesarias más transferencias de datos (lo que implica un mayor número de llamadas al manejador). Así pues, es necesaria una decisión de compromiso que

nos ayude a determinar cuál es el tamaño idóneo del bloque, considerando por un lado la fragmentación, que disminuye al reducir el tamaño del bloque, y por otro la eficiencia, que aumenta al aumentar el tamaño del bloque. Ext2 soluciona parte de este problema diferenciando entre bloques lógicos y bloques físicos. En principio, la asignación de bloques de datos a los Archivos se realiza siempre en forma de bloques lógicos (un bloque lógico tiene un tamaño múltiplo del tamaño del bloque físico, y es transferible físicamente con una única llamada al manejador). Si el último bloque lógico no se llena con los datos del Archivo, entonces se pueden asignar fragmentos del bloque lógico (de tamaño `FragmentSize`, múltiplo a su vez del tamaño del bloque físico).

La relación general entre tamaños es la siguiente:

$$\text{Tamaño del bloque físico} \leq \text{FragmentSize} \leq \text{Tamaño del bloque lógico}$$

Los accesos al disco se realizan, siempre que sea posible, transfiriendo bloques lógicos. Sólo los últimos datos de cada Archivo se asignan en forma de fragmentos para minimizar la fragmentación. Si el Archivo crece, los fragmentos se pueden agrupar para formar bloques lógicos.

## I-NODOS

El i-nodo o nodo índice es la estructura de datos básico empleada por el sistema de Archivos Ext2 para localizar la información relativa a cada Archivo o directorio. Cada grupo mantiene todos los i-nodos en una tabla conocida como tabla de i-nodos. Los i-nodos contienen toda la información acerca del Archivo o directorio que representan.

## DIRECTORIOS

En lo que concierne a los directorios, éstos no son más que archivos normales y corrientes conteniendo información que permite la localización de otros Archivos. Toda la información contenida en un directorio se almacena realmente en un bloque de datos, no existiendo ninguna estructura de datos específica reservada al efecto (al contrario que en otros sistemas de Archivos).

En Linux existe una estructura de directorios que difiere poco de una distribución a otra. Entre los directorios más comunes tenemos:

- `/` : directorio raíz. De él cuelgan todos los demás directorios.
- `/bin` : archivos ejecutables, comandos de usuario.
- `/dev` : archivos de dispositivos (discos, terminales, etc.).
- `/etc` : archivos de configuración, administración e información del sistema.
- `/floppy` : punto de montaje de disquetes.
- `/home` : archivos de usuarios.
- `/lib` : archivos de bibliotecas de desarrollo y material de apoyo.
- `/mnt` : punto de montaje de dispositivos externos.
- `/sbin` : archivos ejecutables de administración.
- `/tmp` : Archivos temporales o zona de trabajo de algunos programas UNIX.
- `/usr` : archivos ejecutables, documentación, referencia.
- `/var` : archivos log y auxiliares.

## MONTAJE DE UN SISTEMA DE ARCHIVOS

Como administradores del sistema, podemos ejecutar una operación de montaje de un sistema de archivos, como la siguiente:

```
mount /dev/hda1 /mnt/disco
```

Con esta orden estamos indicando que queremos montar un sistema de archivos que reside en la partición `/dev/hda1`, en el directorio `/mnt/disco`.

## SEGURIDAD Y PROTECCIÓN

La seguridad en el sistema de archivos Ext2 viene implementada mediante los permisos de acceso a los distintos Archivos y directorios del sistema, así como la comprobación de usuario al iniciar una sesión en un sistema operativo que implemente Ext2.

Al iniciar una sesión, se le indica al usuario que introduzca su login y su password, comprobando el sistema si es un usuario autorizado del mismo. Si no fuera así, el sistema no permitirá la entrada. Esta situación es propiciada por la primera medida de seguridad y protección que se debe tomar: no permitir jamás accesos no autorizados al sistema. Ext2 implementa esta medida mediante un sistema de cuotas de disco y permisos. Todo usuario debe tener una cuenta abierta en el sistema para poder acceder a él, y al iniciar la sesión, el sistema comprueba el login y password almacenados en los archivos relacionados con la gestión de cuentas de usuario, que son creadas por el administrador del



sistema o superusuario.

La seguridad de Archivos y directorios viene dada por los diferentes permisos que se pueden asignar. Existen diversas características que pueden ser asignadas a los Archivos, tales como lectura, escritura, renombrado, oculto, etc. Estas características son las que consiguen que los demás usuarios puedan acceder a los Archivos o no, mediante su asignación o denegación.

## Anexo 7. D: FILE SYSTEM HPFS

### INTRODUCCIÓN A HPFS

El sistema de archivos HPFS (High Performance File System) se presentó por primera vez con OS/2 1.2 para permitir mejor acceso a las unidades de disco duro grandes que empezaban a aparecer en el mercado. Adicionalmente, era necesario un nuevo sistema de archivos para ampliar el sistema de nombres, la organización y la seguridad de las demandas crecientes del mercado de los servidores de red. HPFS mantiene la organización de directorios de FAT, pero agrega la ordenación automática del directorio por nombres de archivo.

Los nombres de archivo se amplían hasta 254 caracteres de doble byte. HPFS también permite que un archivo esté compuesto de “datos” y de atributos especiales que permiten una mayor flexibilidad en cuanto a la compatibilidad con otras convenciones de nomenclatura y con la seguridad. Además, la unidad de asignación se cambia de clústeres a sectores físicos (512 bytes), lo que reduce la pérdida de espacio en disco.

En HPFS, las entradas de directorio contienen más información que en FAT. Además de los atributos de archivo, incluye información acerca de la fecha y hora de modificación, creación y acceso. En lugar de señalar al primer clúster del archivo, las entradas de directorio en HPFS señalan a FNODE.

FNODE puede contener los datos del archivo o punteros que pueden señalar a los datos del archivo o a otras estructuras que, eventualmente, señalan a los datos del archivo.

HPFS intenta asignar en sectores contiguos tantas partes del archivo como sea posible. Esto se realiza para aumentar la velocidad cuando un archivo se procesa secuencialmente.

HPFS organiza una unidad en conjuntos de bandas de 8 MB y, siempre que es posible, un archivo se encuentra en una de estas bandas. Entre cada banda hay mapas de bits de asignación de 2 K, que realizan un seguimiento de qué sectores están asignados dentro de una banda. El sistema de bandas incrementa el rendimiento porque el cabezal de la unidad no tiene que volver al principio lógico (normalmente, el cilindro 0) del disco, sino al mapa de bits de asignación de bandas más cercano para determinar dónde está almacenado un archivo.

Además, HPFS incluye dos objetos de datos exclusivos especiales:

### SUPERBLOQUE

El superbloque se encuentra en el sector 16 lógico y contiene un puntero al FNODE del directorio raíz. Uno de los mayores peligros de HPFS es que si se pierde o se daña el superbloque debido a un sector defectuoso, también se pierde el contenido de la partición, incluso si el resto de la unidad está bien. Sería posible recuperar los datos de la unidad si se copia todo a otra unidad con un sector 16 correcto y se reconstruye el superbloque. Sin embargo, esta tarea es muy compleja.

### BLOQUE DE REPUESTO

El bloque de repuesto se encuentra en el sector 17 lógico y contiene una tabla de “correcciones activas” y el bloque de directorio de repuesto. En HPFS, cuando se detecta un sector defectuoso, se utiliza la entrada de “correcciones activas” para apuntar de forma lógica a un sector correcto existente en lugar de apuntar al sector defectuoso. Esta técnica de tratamiento de errores de escritura se denomina corrección activa.

La corrección activa es una técnica mediante la cual, si se produce un error debido a un sector defectuoso, el sistema de archivos mueve la información a otro sector y marca el sector original como defectuoso. Esto se realiza de forma transparente para cualquier aplicación que efectúe operaciones de entrada y salida de disco (es decir, la aplicación nunca sabe que ha habido problemas en el disco duro). El uso de un sistema de archivos que admite la corrección activa elimina los mensajes de error como “¿Anular, reintentar o ignorar?” de FAT cuando se encuentra un sector defectuoso.

Nota: La versión de HPFS incluida con Windows NT no admite la corrección activa.

### ESTRUCTURA DE UN VOLUMEN HPFS

Los volúmenes HPFS son un nuevo tipo de partición y pueden coexistir en un disco duro junto a otras particiones FAT existentes. Los volúmenes compatibles HPFS tienen un tamaño de sector de 512 bytes y un tamaño máximo de 2199 GiB.

Un volumen HPFS tiene muy pocas estructuras fijas. Los sectores numerados de 0 a 15 de un

volumen (8KiBy) forman el "BootBlock" (bloque de arranque) y contienen un nombre de volumen, un identificador de volumen de 32 bits y un programa de arranque del sistema. El arranque es relativamente sofisticado y puede usar HPFS de un modo restringido para localizar y leer los archivos del sistema operativo donde quiera que éstos se encuentren.

Los sectores 16 y 17 se conocen como el "SuperBlock" y el "SpareBlock" respectivamente. El SuperBlock sólo se modifica por utilidades de mantenimiento y contiene punteros a los mapas de bits del espacio libre, la lista de bloques erróneos, la banda de bloques de directorio y el directorio raíz.

El "SpareBlock" contiene algunas variables y punteros que se verán más adelante; se modifica, aunque infrecuentemente, mientras el sistema se ejecuta.

El resto del disco se divide en bandas de 8Mb. Cada banda tiene su propio mapa de bits de espacio libre en el cual cada bit representa un sector. Un bit a 0 indica que el sector está en uso y un 1 indica que está libre. Los mapas de bits se localizan en la cabeza o en la cola de cada banda de tal forma que dos mapas de bits son adyacentes entre dos bandas consecutivas.

Esto permite que el máximo espacio contiguo que puede asignarse a un archivo sea de 16Mb. Una banda, localizada hacia el centro de búsqueda del disco, es la llamada "banda de bloques de directorios" y recibe especial atención. Nótese que el tamaño de las bandas es una característica de la implementación actual y puede ser cambiada en próximas versiones del sistema de Archivos.

## Anexo 7.E: El Network File System de SUN (NFS)

- NFS = Sistema de archivos distribuidos.
- Idea de base = poder compartir entre varios clientes y servidores heterogéneos un mismo sistema de archivos.
- Una maquina puede ser a la vez cliente y servidor.
- Un servidor NFS exporta sus directorios para que estos sean accesibles por los clientes.
- Si un directorio es exportado, es todo el sub-árbol que es exportado.
- Lista de directorios exportados en /etc/exports.

## ARQUITECTURA

- Un cliente que quiere acceder a un directorio remoto lo debe montar en su propia jerarquía.
- Una estación cliente sin disco (diskless) puede hacer "como si" ella tuviera un disco montando sistemas remotos.
- Una estación con un disco local tendrá una jerarquía en parte local y en parte remota.

Para los programas del cliente:

- no existe diferencia entre archivos locales o remotos.
- Si dos clientes han montado el mismo directorio, ellos comparten los archivos.
- Simplicidad de NFS.

## PROTOCOLOS

NFS debe soportar sistemas heterogéneos (clientes DOS utilizando procesadores Intel, servidores corriendo sobre SUN Sparc, ...).

- Clientes y servidores utilizando diferentes OS y diferentes maquinas.

Es imperativo definir una buena interfase cliente / servidor.

Ventaja de una interfase claramente definida: posibilidad de escribir nuevos clientes y servidores compatibles.

2 protocolos están definidos.

1 protocolo para el mounting y 1 protocolo para el directorio y el acceso a los archivos.

## MOUNTING

Sea C el cliente y S el servidor.

- C envía a S un camino de acceso (el nombre del directorio a montar) y pide permiso para montar el directorio en su maquina.
- El lugar donde C va a montar la directorio no es importante para S.
- Si el camino de acceso es correcto y si el directorio se encuentra en /etc/exports, S reenvía un file handle a C.

El handle esta compuesto por:

- El tipo del sistema de archivos;
- El disco;
- El numero de i-nodo del directorio;
- Información de seguridad (derechos de acceso).

Para leer o escribir en el directorio montada, hay que utilizar este handle.

- Un cliente puede montar directorios sin intervención humana.
- Estos clientes tienen un archivo `/etc/rc` shell script que contiene los comandos de mount lanzados automáticamente durante el boot.
- Denominamos a este proceso, static mounting.
- Las versiones más recientes de SUN UNIX tienen el automounting:
- Directorios remotos son relacionados a directorios locales, pero estos no son montados y sus servidores no son contactados durante el boot.
- La primera vez que un cliente accede a un archivo remoto, los servidores son contactados. El primero que responde gana.

## AUTOMOUNTING vs STATIC MOUNTING

- Ventajas del automounting sobre el static mounting:
- Si un de los servidores NFS nombrado en `/etc/rc` está apagado es difícil de conectar al cliente;
- En el static mounting, no contactamos más que un servidor para cada directorio, mientras que en el automounting podemos contactar varios, esto es tolerancia a las fallas.
- Inconveniente: todos los servidores "alternativos" para un mismo directorio deben ser coherentes, sobretodo utilizado por sistemas de archivos read-only.

## DIRECTORIO Y ACCESO A LOS ARCHIVOS

- 2do protocolo.
- Los clientes envían mensajes para manipular los directorios, leer y escribir archivos y sus atributos (talla, fecha de modificación, propietario, etc.).
- Todas las llamadas a sistemas con tomadas por el NFS, salvo OPEN y CLOSE.
- OPEN y CLOSE no son útiles:
- Para cada operación read o write, el cliente primero envía un pedido LOOKUP que reenvía una fila handle, el servidor no lleva un control de esta demanda;
- Una operación read o write es acompañada del handle.

Si el servidor tiene problemas

- ninguna información acerca de los archivos abierta es perdida (puesto que no existen).
- Un servidor NFS es stateless.

## PROBLEMAS

Un archivo UNIX puede ser abierto y cerrado (locked) para impedir que los otros procesos lo utilicen.

- Archivo cerrado del tipo cerrojos relajados.
- NFS es stateless, no podemos asociar cerrojos a la apertura de un archivo.
- Hace falta un mecanismo externo al NFS para controlar el bloqueo.
- NFS utiliza de igual forma el sistema de protección UNIX (bits rwx para el dueño, grupo y otros).

PERO: el servidor NFS siempre cree al cliente para validar un acceso.

- Qué hacer si el cliente miente?
- Utilización de la criptografía para validar los pedidos.
- Problema: los datos en sí, no están encriptados.
- Las llaves están controladas por el NIS (Network Information Service, o yellow pages).

## MOUNT

- El sysop envía `mount + directorio remoto + directorio local + otros`;
- El programa `mount` recorre el nombre de la dirección remota y encuentra el nombre de la maquina remota asociada;
- `mount` contacta la maquina y pide un handle para este directorio;
- El servidor envía el handle si el pedido esta correcto;
- `mount` hace un llamado al sistema MOUNT (kernel).

El kernel tiene:

- Él construye un v-nodo para el directorio remoto;
- Pide al cliente NFS crear un r-nodo (i-nodo remoto) en su tabla para el file handle;
- El v-nodo apunta al r-nodo.

## OPEN

- El kernel recorre el nombre del camino de acceso, encuentra el directorio, verifica que este es

- remoto y en el v-nodo del directorio encuentra el puntero sobre el r-nodo;
- El kernel pide al cliente NFS abrir el archivo;
- El cliente NFS recupera el nombre del servidor en el nombre del camino de acceso y un handle;
- El cliente crea un r-nodo y advierte a la VFS que crea un v-nodo apuntando al r-nodo;
- El proceso que llama, recupera un descriptor de archivo, relacionado al v-nodo de la VFS.
- Del lado del servidor, nada se crea.

**READ:**

- El VFS encuentra el v-nodo correspondiente;
- El VFS determina si es local o remoto y cual es el i-nodo o r-nodo a utilizar;
- El cliente NFS envía un comando READ, con el handle + el offset.
- Las transferencias se realizan normalmente a 8ko / 8ko, incluso si menor cantidad de bytes es requerida.
- Automáticamente, desde que el cliente recibe los 8ko requeridos, un nuevo pedido de 8ko es enviado.
- Esto es el read ahead.

**WRITE:**

- Las transferencias también se realizan a 8ko / 8ko.
- Mientras los datos escritos representen menos de 8ko, se acumulan localmente.
- Una vez que el cliente ha escrito 8ko, los 8ko son enviados al servidor.
- Cuando un archivo se cierra, los que quedaban son enviados al servidor.
- Utilización del caching:
- Los clientes tienen dos caches: atributos y datos.
- Problemas de coherencias.

**COHERENCIA DEL CACHE**

- No existe una solución "propia": intentamos reducir el riesgo al máximo, pero sin llegar a evitarlo por completo.
- Un timer es asociado a cada entrada del cache. Cuando el timer expira, la entrada es anulada. Normalmente 3 segundos para los datos y 30 segundos para los atributos.
- Cuando un archivo "caché" es abierto, el servidor es contactado para saber la fecha de la última actualización. Si MAJ es más reciente que la copia, la entrada es anulada.
- Cada 30 segundos un timer expira y todas las entradas sucias son enviadas al servidor.

**Anexo 7.F: WINDOWS NT:**

En la primera versión que lanzó Microsoft de Windows NT venía con soporte para tres tipos de sistemas de archivos: FAT, HPFS, y NTFS. Microsoft originalmente diseñó FAT para el DOS, IBM introdujo HPFS para OS/2, y Microsoft creó NTFS para NT. NTFS es el formato nativo de los NT's, y además soporta FAT y HPFS para poder así migrar "paths" (a pesar de que Microsoft no incluía el soporte para HPFS en el NT 4.0).

El gran acierto de Microsoft con la implementación del NTFS fue poder superar las limitaciones que tenían los otros dos sistemas de archivos compatibles de NT. Por ejemplo NTFS soporta seguridad granular en archivos y directorios, siendo que FAT y HPFS no poseían esa habilidad.

A continuación se enumeran algunas de las aplicaciones en las que se hizo fuerte el NT:

Aplicaciones Cliente / Servidor: (servidores de archivos, servidores de cómputo, servidores de bases de datos)

Aplicaciones científicas y de ingeniería.

Aplicaciones de red para grandes sistemas corporativos.

**Características del NTFS:**

- Capacidad de recuperación:

El fabricante de Windows NT dice que posee la capacidad para recuperar el sistema de posibles caídas del sistema o fallas de disco. En caso de que algo de esto ocurra, NTFS puede reconstruir los volúmenes de disco y volver a un estado consistente o seguro. Esto lo hace mediante el modelo "proceso de transacción" para los cambios del sistema de archivos; cada cambio significativo es tratado como una acción atómica, la cual es efectuada por completo o no es ejecutada. Además, el NTFS usa almacenamiento redundante para los datos

críticos del sistema de archivo, entonces el fallo de un sector de disco no va a causar la pérdida de datos describiendo la estructura y el estado del sistema de archivos.

- Seguridad:  
NTFS utiliza el modelo de Windows NT para reforzar la seguridad. Un archivo abierto es tratado como un objeto con un descriptor de seguridad que define sus propios atributos de seguridad.
- Discos y archivos grandes:  
NTFS soporta discos grandes y también archivos grandes más eficientemente que muchos de los otros sistemas de archivos, incluyendo a FAT.
- Cadenas de datos múltiples:  
El contenido de un archivo es tratado como una cadena de bytes. En NTFS es posible definir múltiples cadenas de datos para un solo archivo. Ejemplo de la utilidad de esto es que Windows NT puede ser usado vía remota por un sistema basado en Macintosh para guardar y devolver archivos. En Macintosh, cada archivo tiene dos componentes: los datos del archivo y la sección que guarda la información de este archivo. Entonces NTFS trata estas dos partes como dos cadenas de datos.
- Facilidad para el indexado general:  
NTFS asocia una colección de atributos para cada archivo. Este juego de descripciones de archivos es organizado como una base de datos relacional, entonces los archivos pueden ser indexados por cualquiera de los atributos.

### Volumen y estructura de archivos del NTFS

NTFS hace uso de los siguientes conceptos de almacenamiento en el disco:

- Sector: Es la unidad física más pequeña de almacenamiento en el disco. El tamaño de los datos en bytes es una potencia de 2 y casi siempre de 512 bytes.
- Cluster: Uno o más sectores contiguos (en la misma pista). El tamaño de los clusters también es una potencia de 2.
- Volumen: Es una partición lógica en el disco, consiste en uno o más clusters y usado por el sistema de archivos para asignar espacio. Un volumen consiste en la información del sistema de archivos, una colección de archivos y además cualquier espacio remanente sin asignar en donde se pueden asignar archivos. El volumen puede ser todo o una parte de un disco o también ser extendido a múltiples discos. Si es utilizado RAID 5 a nivel de hardware o software, el volumen consistirá de listas de múltiples discos. El tamaño máximo de volumen para el NTFS es  $2^{64}$  bytes.

El cluster es la unidad fundamental para la asignación en NTFS, el cual no reconoce sectores.

El uso de los clusters para la asignación hace que el NTFS sea totalmente independiente del tamaño físico de los sectores. Esto hace que NTFS soporte fácilmente discos que no son estándar y que no tienen sectores del tamaño de 512 bytes. La eficiencia está en que el sistema de archivos mantiene la información de cada archivo.

Tabla de partición y tamaños de los clusters

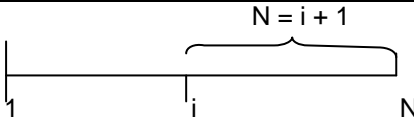

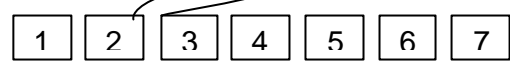
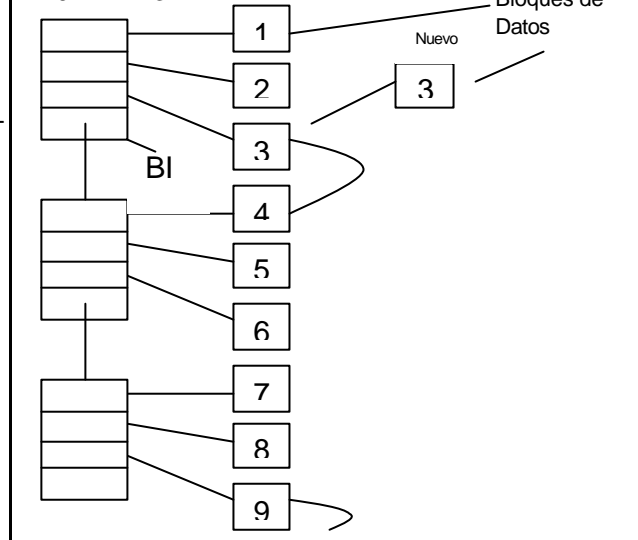
| Tamaño de volumen   | Sectores por cluster | Tamaño del cluster |
|---------------------|----------------------|--------------------|
| $\leq 512$ Mbyte    | 1                    | 512 bytes          |
| 512 Mbyte – 1 Gbyte | 2                    | 1 K                |
| 1 Gbyte – 2 Gbyte   | 4                    | 2 K                |
| 2 Gbyte – 4 Gbyte   | 8                    | 4 K                |
| 4 Gbyte – 8 Gbyte   | 16                   | 8 K                |
| 8 Gbyte – 16 Gbyte  | 32                   | 16 K               |
| 16 Gbyte – 32 Gbyte | 64                   | 32 K               |
| $> 32$ Gbyte        | 128                  | 64 K               |

### Disposición del volumen en el NTFS

Cada elemento en el volumen es un archivo, y cada archivo tiene una colección de atributos. Cada dato contenido en un archivo es tratado como un atributo. Con esta estructura simple, unas pocas funciones generales alcanzan para organizar y manejar el sistema de archivos. Esta disposición consta de cuatro regiones. La primera de estas ocupada por el sector de la partición de booteo, la que contiene la información con respecto a la disposición de volumen y la estructura del

sistema de archivos. Luego de esta sección esta la tabla maestra de archivos (MTF), contiene información de los archivos y carpetas (o directorios). Siguiendo a esta la sección de archivos de sistema.

## Anexo 7.G: Ejemplos para determinar el “costo” de grabar un registro en disco

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Asignación Contigua</b><br> <p>1-Graba el nuevo bloque<br/> 2- Leer grabar desde el i-ésimo anterior hasta el N<br/> <math>\Rightarrow 1 + 2(N - i + 1)</math></p>                                                                                                                                                                                                                                                                                                                         | <b>Ejemplos: insertar el registro 3</b><br>$N=7 \quad i=3$<br> <p>1-Graba el nuevo bloque = 1<br/> Leer y regrabar desde el i-ésimo (inclusive) hasta N<br/> <math>\Rightarrow 2 \times (7-3+1) \Rightarrow 1 + 2 \times (7-3+1) = 1 + 10 = 11</math></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Asignación Enlazada</b><br><p>1- Leer hasta el anterior al i-ésimo <math>\Rightarrow (i - 1)</math><br/> 2- Graba el nuevo<br/> 3- Modifica el <math>(i - 1)</math> ésimo (para alterar al siguiente)<br/> <math>(i - 1) + 1 + 1 \therefore \boxed{i + 1}</math></p>                                                                                                                                                                                                                                                                                                        | <b>Ejemplos: insertar el registro 3</b><br>Para $N=7 \quad i=3$<br> <p>Leer <math>(i - 1) = (3 - 1) = 2</math><br/> Grabar el nuevo = 1<br/> Modificar el <math>(i - 1)</math> ésimo = 1<br/> <math>2 + 1 + 1 = 4</math></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Asignación indexada</b><br><p>1 Cant. de Bloques índices en el arch. = <math>N / (H-i)</math><br/> 2 Cant. de índices por bloque = H<br/> 3 Bloque índice a leer <math>BI = i / (H-i)</math><br/> 4 Leer y modificar el resto de BI <math>\Rightarrow 2(N / (H-i) \lceil - i / (H-i) \rceil) \Rightarrow i / (H-i) + 1 + 1 + 2(N / (H-i) \lceil - i / (H-i) \rceil) + ((N+1)/(H-i) \lceil - (N/H-i) \lceil))</math></p> <p>Graba Nuevo<br/> Modifica BI dónde está el puntero al i-ésimo<br/> Lee y Graba los bloques<br/> Verifica si es necesario agregar un nuevo BI</p> | <b>Ejemplos: insertar el registro 3</b><br>$N=9 \quad i=3 \quad H=4$<br> <p>1er paso: BI a leer = <math>i / (H-i) \lceil = 3/3 \lceil = \leftarrow</math><br/> 2do paso: Cant. de BI a modificar (resto) = <math>2(N / (H-i) \lceil - i / (H-i) \lceil) = 2((9/3) \lceil - (3/3) \lceil) = 2(3-1) = 2</math><br/> <math>\times 2 = \downarrow</math><br/> 3er paso: Grabar el nuevo = <math>\leftarrow</math><br/> 4to. Paso: Modificar el puntero al i-ésimo en el BI = <math>\leftarrow</math><br/> 5to. Paso: Verificar si es necesario grabar un nuevo BI <math>\lceil ((N+1)/(H-i) \lceil - (N/H-i) \lceil)) = ((10/3) \lceil - (9/3) \lceil) = (4-3) = \leftarrow</math> <math>\therefore</math> Si el resultado es 1, hay que asignar un nuevo BI al Archivo. Si es 0 no es necesario, pues el último BI tiene lugar.<br/> Resultado: para este ejemplo el costo de intercalar el registro 3 nuevo es de 8 accesos.</p> |

Ejercicio: Desarrollar el “costo” de borrar un Bloque de datos para los tres casos de estructura de archivos

## AUTOEVALUACIÓN DEL MODULO 7:

### Preguntas:

- 1.- ¿De que depende la organización Lógica y de que la organización Física?
- 2.- ¿Que debe permitir cualquier tipo de organización?
- 3.- ¿Diferencias básicas entre organización secuencial de archivos y organización en pila
- 4.- ¿Que es un I-NODO y cuales son las ventajas del uso del mismo?
- 5.- ¿Es valioso compactar en forma periódica el espacio de almacenamiento del disco?
- 6.- ¿Cual sería la ventaja de que la primera parte de cada archivo UNIX se mantenga dentro del mismo bloque del disco que el i-nodo?
- 7.- ¿Que es y que contiene el Directorio de Archivos?
- 8.- ¿Con que operaciones se cuenta para determinar la posición de lectura en los archivos de acceso aleatorio?
- 9.- ¿En que tipo de organización de archivos se utilizan índices exhaustivos? Y ¿que contienen?
- 10.- Indique los elementos de información de un directorio de archivos para cada archivo del sistema.
- 11.- ¿Qué criterios son importantes en la elección de una organización de archivos?
- 12.- ¿Qué derechos de acceso pueden asignarse a un usuario para un archivo específico?
- 13.-¿Cómo es la estructura física del File System en UNIX?
- 14.- Describa los medios que posee el Sistema Operativo para identificar si una operación es correctamente aplicable a un archivo.
- 15.- ¿Por qué todos los accesos a un archivo son system calls?

### Múltiple Choice:

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1.- El método de listas enlazadas para la asignación del espacio en disco presenta el siguiente inconveniente:</b><br>a) Es necesario conocer el tamaño máximo de archivo en el momento de su creación.<br>b) La compactación resultante de la fragmentación externa.<br>c) El acceso aleatorio a un archivo es extremadamente lento.<br>d) La pérdida de espacio debido a las tablas de índices.<br>e) Todas de las anteriores.<br>f) Ninguna de las anteriores. | <b>2.- En la implementación de archivos...</b><br>a) Mediante asignación contigua no se produce fragmentación interna.<br>b) Mediante asignación no contigua por lista enlazada se favorece el acceso aleatorio.<br>c) Mediante asignación no contigua por tablas de implantación mediante tabla única el número de bloques de un archivo solo está limitado por el tamaño del disco.<br>d) Todas de las anteriores.<br>e) Ninguna de las anteriores son ciertas. |
| <b>3.- Indicar cuales de las siguientes organizaciones de archivos son validas:</b><br>a) Pila.<br>b) Secuencial Indexada.<br>c) Cola.<br>d) Lista enlazada.<br>e) Hashed.<br>f) Todas de las anteriores.<br>g) Ninguna de las anteriores son ciertas.                                                                                                                                                                                                               | <b>4. ¿Cuáles de los siguientes campos se almacenan en el i-nodo de un sistema de archivos UNIX?</b><br>a) Nombre del archivo.<br>b) Permisos de acceso.<br>c) identificador del dueño (owner ID).<br>d) Fechas de acceso y modificación.<br>e) Todas de las anteriores son correctas.<br>f) Ninguna de las anteriores son ciertas.                                                                                                                               |
| <b>5. El Sistema de Gestión de Archivos se ocupa de ...</b><br>a) Llenar el Buffer con los datos o vaciar el mismo.<br>b) Traducir un pedido de acceso desde el espacio lógico de direcciones del archivo a un espacio físico en el soporte.                                                                                                                                                                                                                         | <b>6.- Las Rutinas que permiten la manipulación de los Directorios se invocan mediante ....</b><br>a) Una llamada al Shell efectuadas por el programa del usuario.<br>b) Una llamada al Shell efectuadas por el S.O.                                                                                                                                                                                                                                              |



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>c) Gestionar los Permisos de acceso.</li> <li>d) Llevar la gestión del espacio de almacenamiento en forma automática y transparente</li> <li>e) Todas de las anteriores son correctas.</li> <li>f) Ninguna de las anteriores son ciertas.</li> </ul>                                                                                                                                                                                                                                                                                               | <ul style="list-style-type: none"> <li>c) Una Instrucción específica del programa usuario.</li> <li>d) Una Instrucción especial del Kernel.</li> <li>e) Un System program del compilador.</li> <li>f) El servicio de programas especiales.</li> <li>g) Ninguna de las anteriores corresponde.</li> </ul>                                                                                                                                                                                                                                                                            |
| <b>7.- Los métodos de acceso sirven para ....</b> <ul style="list-style-type: none"> <li>a) Vincular los dispositivos de E/S.</li> <li>b) Acceder a los dispositivos de E/S cuando lo solicita el Programa de Aplicación.</li> <li>c) Manejar las organizaciones de los archivos soportados por el S.O.</li> <li>d) Que el Programador realice detalladamente las operaciones de E/S.</li> <li>e) Realizar las operaciones de E/S por los programas de aplicación.</li> <li>f) Todas las funciones anteriores corresponden.</li> <li>g) Ninguna de las anteriores corresponde.</li> </ul> | <b>8.- La Apertura de archivos (macro open) ...</b> <ul style="list-style-type: none"> <li>a) Se genera mediante una llamada al s.o.</li> <li>b) Genera un cambio de estado del procesador</li> <li>c) Es una instrucción privilegiada</li> <li>d) Describe características del dispositivo a acceder</li> <li>e) Describe características del archivo a acceder</li> <li>f) Evita la asignación dinámica del dispositivo que soporta al archivo a acceder</li> <li>g) Crea archivos</li> <li>h) Ninguna de las anteriores corresponde.</li> </ul>                                  |
| <b>9.- El cierre de archivos (macro close) :</b> <ul style="list-style-type: none"> <li>a) Produce la desasignación del dispositivo.</li> <li>b) Produce la desasignación de las prioridades asignadas al proceso.</li> <li>c) Produce la desasignación del archivo.</li> <li>d) Dispara una interrupción al canal.</li> <li>e) Dispara el llenado inicial de buffers.</li> <li>f) Todas de las anteriores son correctas.</li> <li>g) Ninguna de las anteriores son ciertas.</li> </ul>                                                                                                   | <b>10.- Con respecto a los i-nodos:</b> <ul style="list-style-type: none"> <li>a) Varios nombres de archivos pueden estar asociados a un mismo i-nodo.</li> <li>b) Cada archivo es controlado al menos por un i-nodo.</li> <li>c) Cada i-nodo contiene una referencia del header con datos de un archivo.</li> <li>d) Un i-nodo activo es asociado exactamente con un solo archivo</li> <li>e) Cada archivo es controlado por un i-nodo.</li> <li>f) Todas son verdaderas.</li> <li>g) Ninguna de las anteriores son verdadero</li> </ul>                                           |
| <b>11.- El método de listas enlazadas para la asignación del espacio en disco presenta el siguiente inconveniente...</b> <ul style="list-style-type: none"> <li>a) Es necesario conocer el tamaño máximo de archivo en el momento de su creación.</li> <li>b) La compactación resultante de la fragmentación externa.</li> <li>c) El acceso aleatorio a un archivo es extremadamente lento.</li> <li>d) La pérdida de espacio debido a las tablas de índices.</li> <li>e) Todas de las anteriores son correctas.</li> <li>f) Ninguna de las anteriores.</li> </ul>                        | <b>12.- Marque con una cruz la(s) afirmación(es) correcta(s) sobre File System</b> <ul style="list-style-type: none"> <li>a) Bit Vector permite tanto acceso secuencial como directo.</li> <li>b) La Asignación Contigua solo permite acceso secuencial.</li> <li>c) La Asignación Enlazada es recomendada para el acceso directo.</li> <li>d) Todas las anteriores son correctas.</li> </ul> <p>Ninguna de las anteriores es correcta.</p>                                                                                                                                         |
| <b>13.- En la implementación de archivos:</b> <ul style="list-style-type: none"> <li>a) Mediante asignación contigua no se produce fragmentación interna.</li> <li>b) Mediante asignación no contigua por lista enlazada se favorece el acceso aleatorio.</li> <li>c) Mediante asignación no contigua por tablas de implantación mediante tabla única el número de bloques de un archivo solo está limitado por el tamaño del disco.</li> <li>d) Todas de las anteriores son correctas.</li> <li>e) Ninguna de las anteriores son ciertas.</li> </ul>                                     | <b>14.- La asignación indexada tiene las siguientes ventajas:</b> <ul style="list-style-type: none"> <li>a) No necesita un conjunto contiguo de bloques.</li> <li>b) No necesita leer los bloques previos de disco para acceder a un bloque en particular.</li> <li>c) Posee fragmentación interna y externa.</li> <li>d) Soporta tanto el acceso secuencial como el acceso directo a los archivos.</li> <li>e) Es más eficiente en la lectura de un archivo completo que la asignación contigua.</li> <li>f) Todas las anteriores</li> <li>g) Ninguna de las anteriores</li> </ul> |
| <b>15.- La función de Lectura (Read) de un Archivo se caracteriza por que....</b> <ul style="list-style-type: none"> <li>a) Se libera todo el espacio ocupado por el mismo</li> <li>b) Se realiza mediante una llamada al sistema</li> <li>c) Se debe especificar el nombre del archivo</li> <li>d) No se realiza una Entrada/Salida</li> <li>e) Se necesita el puntero de vinculación</li> <li>f) Se requiere saber en que lugar de la Memoria Central se colocará el siguiente bloque leído.</li> <li>g) Todas las anteriores</li> <li>h) Ninguna de las anteriores</li> </ul>          | <b>16.- Los directorios basados en Grafos Acíclicos...</b> <ul style="list-style-type: none"> <li>a) Fueron diseñado para no compartir información</li> <li>b) Un mismo archivo o un mismo directorio físico solo puede aparecer en el directorio del Dueño.</li> <li>c) Si alguien modifica un archivo que está compartido hay que avisar al otro usuario que se modificó.</li> <li>d) Cuando se borra un archivo compartido, la actualización de los vínculos es muy simple.</li> <li>e) Todas las anteriores</li> <li>f) Ninguna de las anteriores</li> </ul>                    |
| <b>17.- Los directorios se caracterizan por....</b> <ul style="list-style-type: none"> <li>a) Organización de archivos en una unidad de almacenamiento.</li> <li>b) Tabla de símbolos en los que se asocia un nombre simbólico con un archivo.</li> <li>c) Ser un archivo que es gestionado completamente por el S.O.</li> <li>d) Permite establecer una jerarquía de archivos.</li> <li>e) La organización que se usa es secuencial pura y puede estar fragmentado.</li> <li>f) Todas las anteriores</li> <li>g) Ninguna de las anteriores</li> </ul>                                    | <b>18.- El sistema de archivos puede destruirse por fallas...</b> <ul style="list-style-type: none"> <li>a) De hardware.</li> <li>b) En el control de acceso.</li> <li>c) De alimentación de energía eléctrica.</li> <li>d) De protección o seguridad.</li> <li>e) Por problemas con el software del sistema de archivos.</li> <li>f) Todas las anteriores</li> <li>g) Ninguna de las anteriores</li> </ul>                                                                                                                                                                         |
| <b>19.- En la Asignación Indexada (Indexed Allocation) se caracteriza por....</b> <ul style="list-style-type: none"> <li>a) El bloque de índice es un vector de direcciones de bloques libres en el disco.</li> <li>b) El bloque de índice es un vector de direcciones de bloques en el disco.</li> <li>c) El bloque de índice es un vector de direcciones de bloques</li> </ul>                                                                                                                                                                                                          | <b>20.- En la administración el Espacio Libre mediante Mapas de bit...</b> <ul style="list-style-type: none"> <li>a) El sistema enlaza todos los bloques libres del disco.</li> <li>b) Cada bloque del disco se representa con un bit.</li> <li>c) El comienzo de la lista lo marca el puntero Free Space List Head (FSLH).</li> <li>d) Se almacenan las direcciones de n bloques libres en el</li> </ul>                                                                                                                                                                           |

|                                                                                                              |                                                                         |
|--------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| d) ocupados en el disco.<br>La i-ésima entrada en el bloque de índices apunta al i-ésimo bloque del archivo. | e) primer bloque libre disponible en el soporte<br>Todas las anteriores |
| e) La i-ésima entrada en el bloque de índices apunta al primer bloque del archivo.                           | f) Ninguna de las anteriores                                            |
| f) El directorio no contiene la dirección del index block.                                                   |                                                                         |
| g) El directorio contiene la dirección del index block.                                                      |                                                                         |
| h) Todas las anteriores                                                                                      |                                                                         |
| i) Ninguna de las anteriores                                                                                 |                                                                         |

## **Respuestas a las preguntas**

### **1.- ¿De que depende la organización Lógica y de que la organización Física?**

La organización Lógica depende de la forma de acceso a los archivos

La organización Física depende de la estrategia de agrupación y de la estrategia de asignación de archivos.

### **2.- ¿Que debe permitir cualquier tipo de organización ?**

Debe permitir:

- Acceso rápido a la información
- facilidad de actualización de la información
- economía de almacenamiento y mantenimiento sencillo
- fiabilidad para asegurar la confianza en los datos

### **3.- ¿Diferencias básicas entre organización secuencial de archivos y organización en pila**

En la pila:

- la longitud de los registros es variable
- el conjunto de campos es variable
- los registros se ordenan cronológicamente a medida en que los datos van llegando
- No existe una estructura para el archivo de la pila

En los archivos secuenciales:

- la longitud de los registros es fija
- el conjunto de campos es fijo
- los registros se ordenan secuencialmente por la clave (que es un campo que identifica unívocamente al registro)
- Existe una estructura del archivo

### **4.- ¿Que es un i-NODO y cuales son las ventajas del uso del mismo?**

Los i-nodos son estructuras de control que contienen información clave de un archivo, necesaria para el S.O. Puede tener asociados varios nombres de archivos, pero si está activo solo asocia a un único archivo al cual controla.

Las ventajas son las siguientes:

los i-nodos pueden guardarse por bastante tiempo en memoria principal, ya que son de tamaño fijo y relativamente pequeños.

Se puede acceder a un archivo con poca o ninguna indexación, por la cual el acceso es más rápido.

El tamaño máximo de un archivo es lo suficientemente grande para satisfacer a casi toda la aplicación

### **5.- ¿Es valioso compactar en forma periódica el espacio de almacenamiento del disco?**

Si se trata de volúmenes grandes sí lo es, ya que se reduciría la cantidad de espacio desperdiciado. Pero en volúmenes pequeños el tiempo perdido en compactación no se justifica debido a los bajos costos de los mismos.

### **6.- ¿Cual sería la ventaja de que la primera parte de cada archivo UNIX se mantenga dentro del mismo bloque del disco que el i-nodo?**

Generalmente cuando se accede a un determinado archivo se hacen dos lecturas en disco, una al i-nodo del archivo y otra al bloque del archivo. Estos dos accesos generan tiempos de búsqueda bastante grandes. La ventaja radicaría entonces en una considerable reducción de los tiempos de búsqueda, y por lo tanto aumentaría el rendimiento del sistema de archivos.

### **7.- ¿Que es y que contiene el Directorio de Archivos?**

El Directorio es un archivo poseído por el S.O. y accesible a través de diversas rutinas de gestión de archivos. Contiene información sobre los archivos incluyendo atributos, ubicación y propietario. Gran parte de esta información la gestiona el S.O. y en algunos sistemas, parte de la información se almacena en un registro cabecera asociado al archivo, aunque algunos elementos claves deben permanecer en el directorio.

### **8.- ¿Con que operaciones se cuenta para determinar la posición de lectura en los archivos de acceso aleatorio?**

Se cuenta con la operación Read, que da la posición del archivo en la cual iniciar el acceso; y con la operación Seek, que establece la posición de trabajo.

### **9.- ¿En que tipo de organización de archivos se utilizan índices exhaustivos? Y ¿que contienen?**

Se utilizan en archivos indexados, en cuya estructura utilizan múltiples índices por los cuales se accede a los registros. Los índices exhaustivos contienen una entrada para cada registro del archivo principal. Cuando se añade un registro al archivo principal los índices deben actualizarse.

### **10.- Indique los elementos de información de un directorio de archivos para cada archivo del sistema.**

- **Información Básica:** nombre del archivo, tipo de archivo y organización del archivo.
- **Información de Direccionamiento:** volumen, dirección de comienzo, tamaño usado y tamaño asignado.
- **Información de Control de Acceso:** propietario, información de acceso y acciones permitidas.
- **Información de Uso:** fecha de creación, identidad del creador, fecha de la última lectura, identidad del último lector, fecha de última modificación, identidad del último modificador, fecha de la última copia de seguridad y utilización actual.

### 11.- ¿Qué criterios son importantes en la elección de una organización de archivos?

Los criterios más importantes son los siguientes:

- Acceso rápido para la recuperación eficaz de información.
- Facilidad de actualización para ayudar a mantener la información al día.
- Economía de almacenamiento para reducir costos.
- Mantenimiento sencillo para reducir costos y la posibilidad de errores.
- Fiabilidad para asegurar la confianza en los datos.

### 12.- ¿Qué derechos de acceso pueden asignarse a un usuario para un archivo específico?

Derechos de Acceso:

El sistema de archivos provee una herramienta flexible para permitir compartir extensos archivos entre los usuarios. El sistema de archivos debe proporcionar un número de opciones de modo en que un archivo que es accedido pueda ser controlado. Normalmente, al usuario o a los grupos de usuarios se les otorgan ciertos derechos de acceso a cada archivo. Un amplio rango de derechos de acceso se ha venido usando. La siguiente lista representa los derechos de acceso que pueden ser asignados a un usuario en particular para un archivo en particular:

- Ninguno: El usuario no puede siquiera determinar la existencia del archivo ni mucho menos acceder al mismo. No se permite al usuario leer el directorio de usuario que incluya al archivo.
- Conocimiento: El usuario sabe de la existencia del archivo y quien el dueño. El usuario puede solicitar los derechos de acceso adicionales al propietario.
- Ejecución: El usuario puede ejecutar y cargar un programa pero no copiarlo.
- Lectura: El usuario puede leer el archivo para cualquier propósito, incluyendo copia y ejecución.
- Adición: El usuario puede añadir datos al archivo, generalmente al final, pero no puede modificar o borrar el contenido del mismo.
- Actualización: El usuario puede modificar, borrar y añadir otros datos al archivo.
- Cambio de protección: El usuario puede cambiar los derechos de acceso otorgados a usuarios.
- Borrado: El usuario puede borrar el archivo del sistema de archivos.

Los derechos constituyen una jerarquía. Si un usuario adquiere el derecho de la actualización para un archivo determinado, también habrá adquirido los derechos siguientes: conocimiento, ejecución, lectura y adición.

El propietario de un archivo dispone de los derechos de acceso listados antes y puede otorgar derechos a los otros. Puede ofrecerse acceso a las siguientes clases de usuarios:

- Usuario específico: Usuarios individuales quienes son designados por su ID de usuario.
- Grupos de usuarios: Un conjunto de usuarios no definidos individualmente.
- Todos: Todos los usuarios que tengan acceso al sistema. Estos serán archivos públicos.

#### Accesos Simultáneos:

Cuando el acceso es concedido para añadir o actualizar un archivo a más de un usuario, el sistema operativo o el sistema de gestión de archivos debe hacer cumplir una disciplina. Un método de fuerza bruta consiste en permitir a los usuarios bloquear el archivo entero cuando lo vaya a actualizar. Un mejor control es bloquear los registros individuales durante la actualización. Al disertar la posibilidad de accesos comparados, deben abordarse aspectos de mutua exclusión e interbloqueo.

### 13.-¿Cómo es la estructura física del File System en UNIX?

Un File System es una colección de Bloques contiguos en un dispositivo de almacenamiento. Cada File System debe contener los siguientes componentes:

- **Boot Block:** Es el primer block del File System (Block 0). Muchas veces contiene el programa de arranque (booteo) con el cual se inicializa el S.O..
- **Super Block:** Los datos específicos del File System y sobre él, son almacenados en el Super Block. Es el segundo Block (Block 1) en el File System.
- **List Blocks:** Tienen un Inodo por archivo. Cada Archivo tiene un inodo que está almacenado en el File System. Varios Inodos puede incluir el File System Block (hasta un máximo de 99). Los bloques de la I-List comienzan en el Block 2 y finalizan en la locación que se especifica (indirectamente) cuando se crea el File System.
- **Data Blocks:** Los bloques de datos son los datos definidos por el usuario ya sea previamente o actualmente. Los bloques de datos es la sección más grande del File System.

### 14.- Describa los medios que posee el Sistema Operativo para identificar si una

**operación es correctamente aplicable a un archivo.**

El tipo de archivo, que generalmente esta representado por su Bloque de Control (File Control Block), le confiere al archivo una determinada estructura. El Bloque de Control del archivo contiene una serie de propiedades con información de utilidad para su utilización y manipulación, como la longitud, hora de creación, tipo, permisos de cada usuario, etc.

Esta estructura tiene que ser primero reconocida por el sistema operativo, se dice que el sistema operativo soporta la estructura. De esta manera se asegura que se pueda operar correctamente con ese tipo de archivo y luego verificar si los permisos de accesos están autorizados (open) y por último verificar que operaciones (R, W, X, etc.) están permitidas para ese acceso.

**15.- ¿Por qué todos los accesos a un archivo son system calls?**

Todos los accesos a un archivo son system calls porque son recursos compartidos.

***Respuestas del múltiple choice.***

- |         |         |                 |               |                  |
|---------|---------|-----------------|---------------|------------------|
| 1.- a.  | 2.- e.  | 3.- a, b, d, e. | 4.- e.        | 5.- b, d.        |
| 6.- d.  | 7.- c.  | 8.- a, b, c.    | 9.- c.        | 10.- a, c, d, e. |
| 11.- c. | 12.- e. | 13.- e.         | 14.- a, b, d. | 15.- b, c, e, f. |
| 16.- f. | 17.- f. | 18.- a, c, e.   | 19.- b, d, g. | 20.- b.          |