CAPITULO 15

ADMINISTRACION DE LA INFORMACION

15.1 - **INTRODUCCION**

Ya hemos visto con anterioridad el ciclo por el cual pasa todo trabajo desde que es submitido a un sistema de cómputo, hasta que eventualmente termina toda actividad asociada a este.

Ahora bien, qué implica poder llevar a cabo todos los objetivos presentes en un trabajo o tarea?. Obviamente, realizar todos los procesos pertenecientes al Trabajo en cuestión, para lo cual es necesario en el momento en que uno de dichos procesos se encuentra en estado de ejecutando tener disponibilidad de acceso a datos y/o programas que constituyen parte del proceso.

Es claro que no necesariamente toda esta información se encontrará en memoria al mismo tiempo, y tampoco necesariamente en el momento en que es reclamada por el procesador. Además, qué pasa con todo aquel conjunto de información que constituye el resto de trabajos que se encuentra en ese preciso momento en el sistema de cómputo?. Toda esta información se encuentra en lo que se da en llamar memoria secundaria o almacenamiento secundario (discos, cintas, etc.).

Por todo lo anteriormente expuesto, es claro que el sistema operativo debe constar de un módulo que se encargue del manejo de aspectos tales como el almacenamiento y recuperación de la información que se encuentra dentro del sistema. Resumiendo, podemos considerar que el Administrador de Información posee las siguientes funciones básicas:

1. Llevar rastro de toda la información en el sistema a través de varias tablas, siendo la mayor de ellas el Directorio de Archivos.

   Estas tablas contienen para cada archivo:

   - el nombre,
   - ubicación,
   - longitud (cantidad de registros del archivo),
   - longitud del registro lógico,
   - longitud del registro físico,
   - formato de registros (fijo, variable, etc.),
   - organización (secuencial, indexada, al azar, etc.),
   - fecha de creación,
   - fecha de expiración,
   - derechos de acceso,
   - extensiones.

   Las extensiones de un archivo obedecen al hecho de que a menudo no es posible almacenar el mismo en un fragmento contiguo del periférico. Por lo tanto en el campo de extensiones se indican la cantidad de fragmentos que existen de ese archivo y se inicia a partir de la primera extensión (usualmente denominada alocación primaria del archivo) una cadena que apunta a las otras entradas del Directorio en donde continúan los otros pedazos del archivo. En adecuación a esto el tamaño total del archivo es el de su alocación primaria más el tamaño de todas sus extensiones.
2. Decidir que política es utilizada para determinar dónde y cómo la información es almacenada. Algunos factores que influyen en esta política son:
   - utilización efectiva del almacenamiento secundario,
   - acceso eficiente,
   - flexibilidad a usuarios, y por último
   - protección de derechos de acceso sobre información pedida.
3. Asignación del recurso de la información. Una vez que la decisión de permitir a un proceso tener acceso a cierta información es efectuada, los módulos de ubicación de la información deben encontrar dicha información, hacer accesible dicha información al proceso y por último establecer los derechos de acceso apropiados.
4. Desasignación del recurso de la información. Una vez que la información no es necesitada más, las entradas en tablas asociadas a esta información pueden ser eliminadas. Si el usuario ha actualizado la información, la copia original de la información puede ser actualizada para posible uso de otros procesos.

Comúnmente al conjunto de módulos del Administrador de Información se lo referencia como el *Sistema de Archivos*. Es una meta al concebir un Sistema de Archivos que el mismo libere al usuario de problemas tales como la ubicación de la información que está referenciando, así como el formato real de dicha información en el sistema y el problema de acceso de E/S. Al quedar liberado el usuario-programador de tales escollos, puede concentrarse totalmente en su problema específico (estructura lógica y operaciones necesarias para procesar dicha estructura).

15.2 - **MODELO GENERAL DE UN SISTEMA DE ARCHIVOS**

Para el Sistema de Archivos que tomaremos como ejemplo en este capítulo, asumiremos siempre las posibilidades más sencillas. No obstante, a medida que se vayan explicando los distintos niveles de la implementación, se dejará constancia de sus deficiencias y posibilidades de mejora.

Consideraremos que todos los módulos de la implementación se encuentran dispuestos en niveles o capas, de tal forma que dado un módulo, este sólo depende de aquellos módulos que se encuentran en niveles inferiores a él y sólo efectúa llamadas hacia estos módulos. Son obvio las ventajas en cuanto a modularidad se refiere el concebir el Sistema de Archivos como una estructura jerárquica de este tipo. En la figura 15.1 podemos ver dicho modelo.

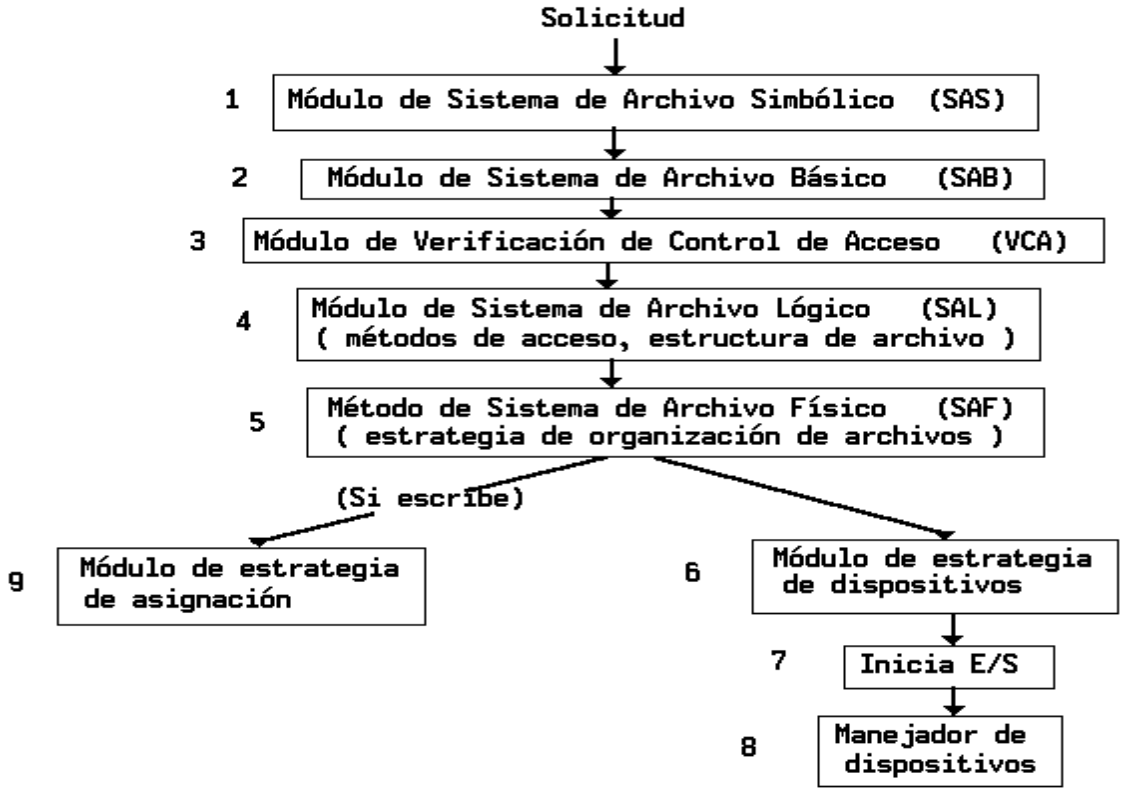


Fig. 15.1. - Modelo jerárquico de un sistema de archivos.

También es necesario destacar que aunque los detalles específicos dados a continuación pueden variar significativamente de un sistema a otro, la estructura básica es común a la mayoría de los sistemas actuales. Es decir, dependiendo del sistema específico que tratemos, algunos de estos módulos se fusionan, se desglosan en aún más módulos, o incluso algunos desaparecen. Aún así la estructura subyacente sigue siendo la misma.

15.2.1 - **ESTRUCTURA Y MANTENIMIENTO DEL DIRECTORIO DE ARCHIVOS**

Antes de describir el funcionamiento de los distintos módulos que conforman el Sistema de Archivos es necesario sentar algunas bases acerca de los siguientes ítems:

Estructura del Directorio de Archivos:

Nosotros visualizaremos al Directorio de Archivos como una tabla donde cada entrada en ella corresponde a un archivo. Definiendo a un archivo como una colección de unidades de información relacionadas llamadas registros.

Así, el contenido de una entrada del Directorio de Archivos de nuestro ejemplo contendrá los siguientes datos:

- Número de entrada,
- Longitud de registro lógico,
- Cantidad de registros lógicos,
- Longitud del registro físico,
- Formato de los registros,
- Organización,
- Dirección al primer bloque físico,
- Protección y control de acceso.

El Número de entrada en la tabla: se mantiene dicho número pues este se convertirá en el identificador del archivo dentro del sistema.

Desde el 2do al 7mo ítem de la tabla sirve para poder ubicar y mapear un registro lógico en su verdadera dirección física en el almacenamiento secundario.

La Protección y control de acceso indica qué usuario tiene derecho de acceso sobre el archivo y que tipo de operaciones se pueden realizar sobre el archivo (solo lectura, lectura/escritura, etc.).

Creación de una entrada del Directorio de Archivos:

Los comandos CREAR y BORRAR incorporan o eliminan respectivamente una entrada en el Directorio de Archivos. Así, una posible sintaxis del comando CREAR sería:

CREAR nombre-archivo, longitud-registro-lógico, cantidad-registros-lógicos, [ubicación-primer-bloque-físico], [formato], [organización]

Ubicación del Directorio de Archivos:

Si todo el Directorio de Archivos fuera mantenido todo el tiempo en memoria principal, se necesitaría una gran cantidad de memoria sólo para este propósito, por lo que surge la idea de mantener al Directorio de Archivos como un archivo dentro de un volumen de almacenamiento (cinta, diskette, disco, etc.).

No obstante se soluciona el problema del desperdicio de memoria principal, ahora nos encontramos ante el inconveniente de que la búsqueda de una entrada en dicho directorio puede ser considerablemente larga, si este constara, por ejemplo, de unas cuantas miles de entradas.

Este problema se soluciona si mantenemos en memoria aquellas entradas del Directorio que pertenecen a archivos que fueron referenciados anteriormente. En llamadas subsiguientes no habrá desperdicio de tiempo de E/S.

Muchos sistemas operativos constan de pedidos especiales tales como ABRIR y CERRAR (Open y Close respectivamente) un archivo determinado. ABRIR coloca en memoria la entrada en la tabla perteneciente al archivo en cuestión y CERRAR constituye su contrapartida.

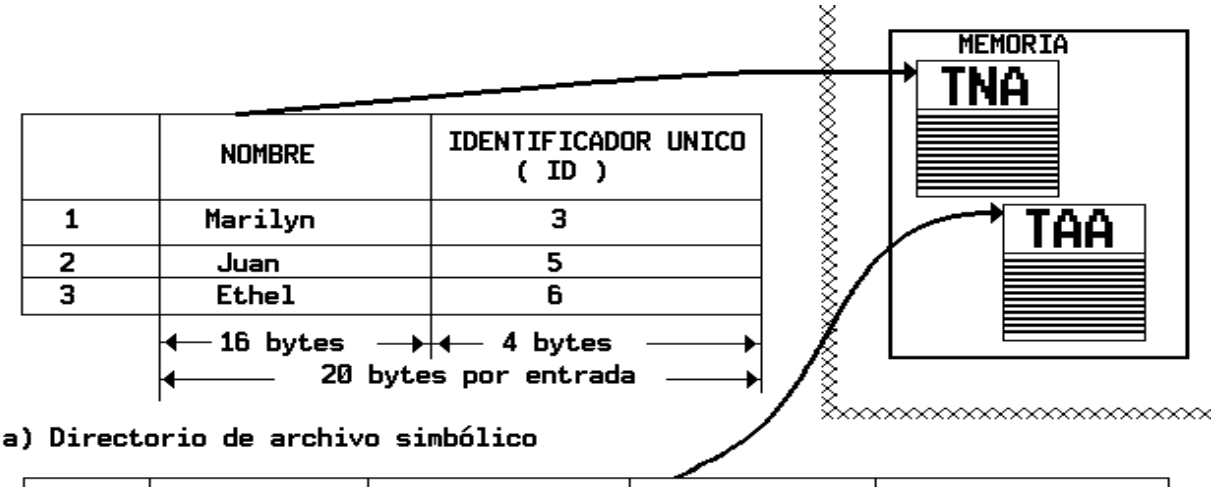
15.2.2 - **SISTEMA DE ARCHIVO SIMBOLICO**

El primer módulo que es llamado cuando hacemos un pedido al Sistema de Archivos es el módulo denominado Sistema de Archivo Simbólico (SAS).

Una típica llamada sería por ejemplo:

CALL SAS(READ,"JUAN",4,1200)

Donde estamos pidiendo que se lea el registro lógico número 4 del archivo "JUAN", para colocar su



	NOMBRE	IDENTIFICADOR UNICO (ID)
1	Marilyn	3
2	Juan	5
3	Ethel	6

16 bytes — 4 bytes

20 bytes por entrada

a) Directorio de archivo simbólico

ID	LONGITUD DE REGISTRO LOGICO	CANTIDAD DE REGISTROS LOGICOS	DIRECCION AL PRIMER BLOQUE FISICO	DIRECTORIO Y CONTROL DE ACCESO
1	---	---	0	Dir. Básico
2	20	3	1	Dir. Simbólico
3	80	10	2	Todos Leen
4	1000	3	3	Libre
5	500	7	6	Todos Leen
6	100	30	12	MARTA Lee JUAN Lee/Graba
7	1000	2	10	Libre
8	1000	1	15	Libre

b) Directorio de archivo básico

Fig. 15.2. - Directorios Simbólico y Básico.

contenido en la dirección 1200 de memoria principal.

El Sistema de Archivo Simbólico usa el nombre del archivo para localizar la única entrada que posee el archivo en el Directorio de Archivos.

Ahora bien, si supusiéramos que existe una correspondencia biunívoca entre el nombre que puede tener un archivo y el archivo en cuestión, nos encontraríamos que el sistema adolece de una gran falla. Pues no podríamos referenciar a un archivo por diferentes nombres y distintos archivos no podrían tener el mismo nombre.

Para implementar tal facilidad dividimos el Directorio de Archivos en dos partes. Un Directorio de Archivo Simbólico (DAS) y un Directorio de Archivo Básico (DAB) como se muestra en la figura 15.2.

El Sistema de Archivo Simbólico debe buscar en el Directorio de Archivo Simbólico la entrada perteneciente al archivo requerido y de esta forma encontrar su único identificador (ID) dentro del sistema, para pasárselo al módulo denominado Sistema de Archivo Básico (SAB).

En nuestro ejemplo CALL SAS(READ,"JUAN",4,1200) devolvería 5 donde 5 es el ID de del archivo "JUAN".

Normalmente como el DAS es mantenido en el periférico de almacenamiento, lo que se hace es copiar en memoria las entradas del DAS que corresponden a archivos en uso (llamados archivos abiertos o activos), de tal forma que estas entradas son usadas para evitar E/S sobre la misma zona en el periférico. Esta tabla que se mantiene en memoria principal con las entradas del DAS correspondientes a archivos abiertos recibe el nombre de Tabla de Nombres Activos (TNA).

15.2.3 - **SISTEMA DE ARCHIVO BASICO**

El segundo módulo que es llamado en la secuencia se denomina Sistema de Archivo Básico (SAB). La llamada sería de esta forma:

CALL SAB(READ,5,4,1200)

Donde todos los parámetros son iguales a la llamada del módulo SAS, a excepción del segundo parámetro que constituye el identificador que le pasó el SAS.

El SAB se vale del identificador del archivo (ID) para localizar la entrada correspondiente a este en el Directorio de Archivo Básico.

Dicha entrada es mantenida en memoria principal con el objeto de ahorrar posteriores E/S buscando la misma entrada. La tabla que alberga todas las entradas del Directorio de Archivo Básico de los archivos abiertos recibe el nombre de Tabla de Archivos Activos (TAA). Esta tabla se genera y mantiene en memoria principal a diferencia del directorio básico (DAB) y del directorio simbólico (DAS). Su entrada consta de la información que puede visualizarse en la Fig. 15.3.

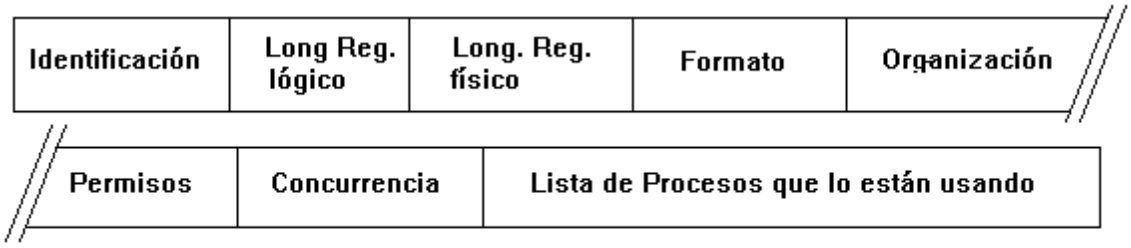


Fig. 15.3. - Tabla de Archivos Activos (TAA).

Para la próxima etapa - Verificación de Control de Acceso (VCA)- utilizaremos en vez del ID del archivo, su entrada correspondiente en la TAA, la cual contiene la información del archivo ID 5.

La invocación al módulo de Verificación de Control de Acceso es de la siguiente forma:

CALL VCA(READ,5,4,1200)

Donde 5 es la entrada 5 de la TAA que corresponde al archivo "JUAN" y los demás son exactamente los mismos parámetros de la llamada al módulo anterior.

En resumen podemos decir que el módulo SAB se encarga de:

1. El Directorio de Archivo Básico.
2. La Tabla de Archivos Activos.
3. La comunicación con el módulo VCA.

15.2.4. - **VERIFICACION DE CONTROL DE ACCESO**

Este módulo actúa como un punto de control entre el Sistema de Archivo Básico y el módulo de Sistema de Archivo Lógico, de tal forma que verifica si la función que se quiere realizar sobre el archivo en cuestión (READ, WRITE, etc.) está explicitada en la entrada correspondiente al archivo en la Tabla de Archivos Activos.

En caso que no sea permitido realizar dicha operación sobre el archivo estamos en presencia de una condición de error, por lo cual el pedido al Sistema de Archivos es abortado. Si efectivamente se puede realizar esa operación entonces el control pasa directamente al módulo de Sistema de Archivo Lógico.

15.2.5. - **SISTEMA DE ARCHIVO LOGICO**

Una llamada al módulo de Sistema de Archivo Lógico (SAL) posee la misma sintaxis que la llamada al módulo anterior. Luego, para el ejemplo que damos quedaría:

CALL SAL(READ,2,4,1200)

El Sistema de Archivo Lógico convierte el pedido de un registro lógico en el pedido de una secuencia de bytes lógicos, la cual se entrega al Sistema de Archivo Físico (SAF).

Esto es así, puesto que para el SAF un archivo no es más que una secuencia de bytes sin ningún tipo de formato.

Para nuestro ejemplo, vamos a suponer un formato de registro lógico de longitud fija, luego la conversión necesaria la podemos obtener de la información de la entrada en la TAA.

Dirección del byte lógico (dbl) = (número de registro - 1) * longitud del registro lógico = 3 * 500 = 1500

long. de la secuencia de bytes lógicos (lsb) = long. del registro lógico

Teniendo ya calculado el comienzo de la secuencia de bytes y su longitud, el SAL invoca al módulo de Sistema de Archivo Físico (SAF):

CALL SAF(READ,2,1500,500,1200) 1500 es dbl y 500 el lsb

Donde el tercer y cuarto par metro corresponden al dbl y el lsb respectivamente y el resto de parámetros son exactamente los mismos que los de la llamada el módulo anterior.

15.2.6. - **SISTEMA DE ARCHIVO FISICO**

El SAF tiene por función determinar en que bloque físico del dispositivo de almacenamiento se encuentra la secuencia de bytes lógicos pasados por el SAL, para lo cual se vale de la entrada en la TAA más el dbl y el lsb.

El bloque es entonces leído y colocado en un buffer preasignado en memoria principal, luego es extraído a partir de este buffer la secuencia de bytes pedidos y colocados en el área de buffer del usuario.

Para realizar estos cálculos el SAF debe saber las funciones de mapeo y longitud del bloque físico que utiliza cada periférico de almacenamiento. Si estas fueran las mismas para todo tipo de periférico podrían estar tranquilamente insertas en las mismas rutinas del SAF, pero tal cosa no sucede pues hay variaciones de un periférico a otro.

Esto se resuelve manteniendo tal información en el mismo volumen de almacenamiento, de tal forma que cuando se inicializa el sistema toda esta información pueda ser leída en una entrada de la Tabla de Archivos Activos. Si se hace un pedido de escritura y el bloque sobre el cual se quiere escribir no está asignado previamente entonces el Sistema de Archivo Físico invoca al Módulo de Estrategia de Asignación (MEA) para que este le proporcione un bloque libre en memoria secundaria sobre el cual pueda efectuar la escritura.

15.2.7. - **MODULO DE ESTRATEGIA DE ASIGNACION**

Este módulo se encarga de llevar un registro del espacio libre disponible en el almacenamiento secundario, tal información aparecerá reflejada en la TAA.

En la figura 15.B se muestra como se puede hacer uso del Directorio de Archivo Básico para llevar cuenta de esta información, es decir tratamos a los espacios libres en memoria secundaria como si fueran archivos. Obviamente existen otras técnicas más óptimas para tratar este problema.

15.2.8 - **MODULO DE ESTRATEGIA DE PERIFERICO**

Este módulo convierte el número de bloque físico en el formato adecuado para el periférico en cuestión (por ejemplo bloque 7 = pista 3 ,sector 23). Además de esto, inicializa los comandos adecuados de E/S para el tipo de periférico sobre el cual se va a realizar la operación.

Cabe destacar que todos los módulos anteriormente citados son totalmente independientes de cualquier tipo de periférico con excepción del último nombrado. De aquí en más, el control pasa a manos del Administrador de Entrada-Salida.

15.2.9. - **Resumen de los módulos**

A guisa de resumen reseñamos brevemente todos los módulos antes mencionados:

1) **Sistema de archivos simbólicos (SAS)**: transforma el nombre del archivo en el identificador único del Directorio de archivos. Utiliza la Tabla de nombres activos (TNA) y el directorio de archivos simbólico (DAS).

2) **Sistema de archivos básico (SAB)**: copia la entrada de la VTOC en memoria. Utiliza el directorio de archivos básicos (DAB) y la Tabla de archivos activos (TAA).
3) **Verificación de control de acceso (VCA)**: verifica los permisos de acceso al archivo.
4) **Sistema de archivo lógico (SAL)**: transforma el pedido lógico en un hilo de bytes lógicos.
5) **Sistema de archivo físico (SAF)**: calcula la dirección física.
6) **Módulo de estrategia de asignación (MEA)**: consigue espacio disponible en el periférico (casos de grabación).
7) **Módulo de estrategia de periférico**: transforma la dirección física según las características exactas del periférico requerido.

15.3. - **ESTRUCTURAS DE DIRECTORIOS**

Al hablar del Sistema de Archivos Básico hemos aclarado que el mismo opera sobre el Directorio de Archivos Básico (DAB). Usualmente este directorio se encuentra almacenado el algún periférico del sistema que posea velocidad alta.

Si pensamos un momento la cantidad de archivos que pueden llegar a existir en cualquier centro de cómputos (aún siendo pequeño) comprenderemos inmediatamente que contar con un único directorio básico en forma de una interminable tabla que deberá ser accedida toda vez que se requiera un archivo, resulta poco práctico.

En tal sentido es conveniente estructurar los directorios en forma jerárquica (similarmente a lo que se realiza con un archivo indexado con diferentes niveles de índices) para proveer una mejor y más óptima forma de acceso, un marco de trabajo ordenado y protección para todos los usuarios en su conjunto.

La cuestión de un marco de trabajo ordenado permite armar estructuras que por ejemplo pueden agrupar en un único directorio todos los archivos de un usuario específico, como así también todos los archivos y/o programas de una aplicación específica (sueldos, cuentas corrientes, etc.). La cuestión de cómo proteger dichas estructuras se verá más adelante (Ver punto 15.4).

15.3.1. - **Directorios de un solo nivel**

Una de las formas de estructurar un directorio puede verse en la Fig. 15.4. Nótese que en este caso la entrada en el directorio de archivos básico está apuntando a una estructura que mapea todos los bloques del archivo real en disco.

La cantidad de acceso necesarios para llegar a un archivo en una estructura de este tipo serán 2, uno para llegar al directorio básico y uno para llegar al archivo real. Si consideramos que para llegar a la tabla de mapas de bloques se requiere un nuevo acceso entonces la cantidad de acceso será del orden de 3.

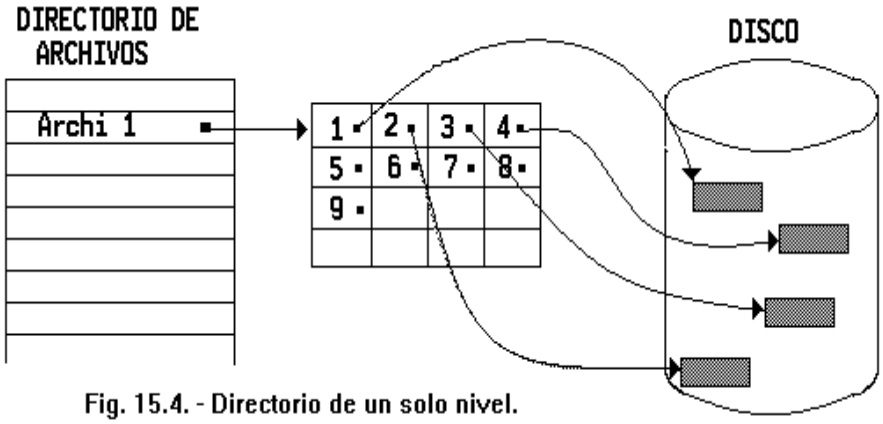


Fig. 15.4. - Directorio de un solo nivel.

La tabla de mapeo se utiliza para optimizar la utilización del espacio en el disco aunque cuenta con el inconveniente de que la información puede estar muy dispersa dentro del dispositivo degradando todo tipo de proceso que requiera leer una gran cantidad de registros de este archivo.

15.3.2. - **Directorios de varios niveles**

Una estructura más compleja de directorios puede visualizarse en la Fig. 15.5. En este caso la cantidad de accesos para llegar a un archivo dependerá de la profundidad de niveles del directorio.

En forma genérica podemos decir que para llegar a una entrada en el nivel n se requerirán no menos de n+1 accesos a disco, debido a que es necesario siempre recorrer los n niveles que apuntan a dicho archivo para finalmente acceder finalmente al archivo en sí.

Nótese en el ejemplo que para llegar al archivo ALFA se deberá ingresar al sistema por el directorio A, de allí pasar al segundo nivel en el directorio AA y de allí acceder al archivo; por lo tanto para proveer el camino completo para referenciarse al archivo un usuario debería especificar el nombre del archivo de la forma A/AA/ALFA pudiendo utilizarse como separadores la /, el ., o cualquier otro similar provisto por el lenguaje de control.

En las estructuras de este tipo debe tenerse presente que en las entradas de los directorios debe existir algún campo que permita diferenciar si la información contenida en esa entrada corresponde a un directorio o a un archivo (por ejemplo la entrada correspondiente al directorio ABA y la entrada correspondiente al archivo ALFA en el directorio AA segundo nivel).

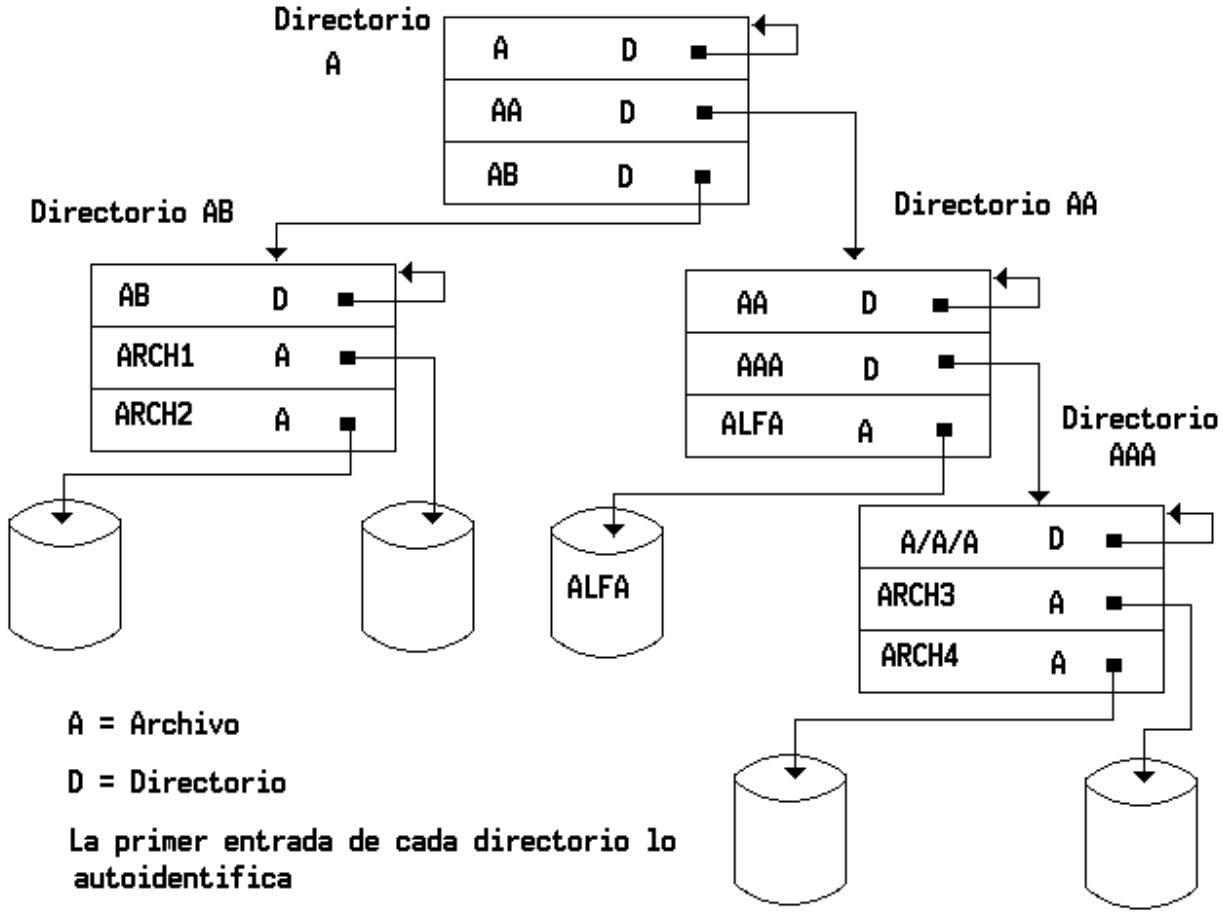


Fig. 15.5. - Directorios de varios niveles.

15.4. - **Estructuras de Control de Acceso**

El control de acceso a un archivo está dado por una serie de permisos que son controlados por el módulo de Verificación de Control de Acceso (VCA).
Tales permisos se almacenan conjuntamente con la información del directorio básico o alguno de sus niveles y pueden constituir asimismo archivos de permisos. Estos datos serán copiados por el Sistema de archivos Básicos en la TAA y luego consultados por el VCA para autorizar el acceso.

Una vez liberado el archivo y de haber ocurrido algún cambio en los permisos la información deberá ser devuelta a su lugar de almacenamiento en memoria secundaria.

15.4.1. **LISTA DE CONTROL DE ACCESO**

El concepto de la Lista de Control de Accesos (LCA) se basa en que para cada archivo en el sistema se asocia una lista de permisos (que puede ser otro archivo) en donde se especifica para cada usuario que tipo de acciones puede realizar sobre ese archivo.

En el ejemplo de la figura 15.6 el usuario JOSE puede Leer sobre el archivo ALFA y el usuario MARIA puede Leer/Grabar sobre el archivo ALFA.

Suele existir también el permiso de Owner que indica el nombre del usuario que originalmente creó ese archivo y que puede realizar todo tipo de acciones sobre él. En el ejemplo no se ha especificado este permiso debido a que para llegar al archivo ALFA se debe ingresar obligatoriamente por el directorio del usuario PEPE asumiéndose entonces que dicho usuario es el Owner.

Pueden existir también permisos de Ejecución cuando se trata de archivos que corresponden a códigos objeto de programas. Este permiso es muy útil ya que solamente permite el acceso a tal archivo si la acción de-

seada es Ejecutar y en cambio lo niega si la acción deseada es de Lectura (previene copias ilegales). La acción se puede diferenciar en virtud de la naturaleza del proceso que la invoca (un COPY es diferente de un EXEC).

Nótese que por la estructura que poseen el directorio de ejemplo es requisito indispensable que para que un usuario llegue a un archivo que no le es propio conozca el nombre del usuario que es su propietario a fin de

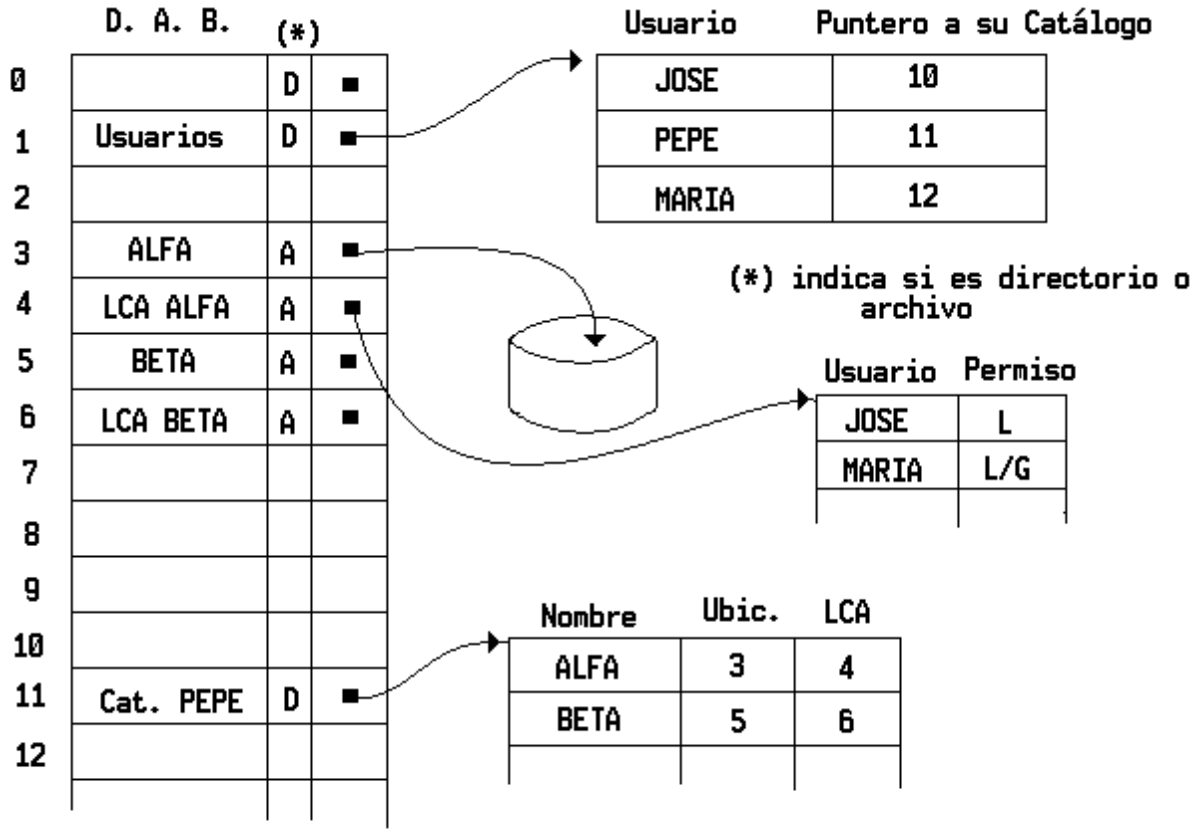


Fig. 15.6. - Sistema de Lista de Control de Acceso (LCA).

proveer la información de la vía necesaria para ubicar ese archivo (path). En tal estructura un acceso al directorio básico con el nombre ALFA no sería útil debido a que el módulo VCA exige la verificación de la LCA del archivo y tal información solamente existe a nivel del directorio del usuario PEPE (obvio, el acceso por defecto es Ninguno).

La forma más sencilla de proteger el directorio básico es aprovechar la entrada cero, que existe siempre en todo tipo de directorio y que sirve para autoidentificarse, para agregar allí la información de donde se encuentra su LCA.

En LCA la capacidad de acceso a un archivo pertenece al mismo archivo.

15.4.2. - **LISTA DE CONTROL DE USUARIO**

A diferencia de la LCA la Lista de Control de Usuario (LCU) se basa en el concepto de que para cada usuario existente en el sistema hay una lista de los archivos a los cuales puede acceder y que acciones puede realizar sobre ellos.

En la Fig. 15.7 podemos ver como con prácticamente la misma estructura de directorios anterior construimos una LCU.

Para cada usuario tenemos una entrada que corresponde al directorio de los archivos que le son propios y un puntero a una lista de archivos que indica la ubicación del archivo en el DAB, su nombre y el permiso de acceso.

Aquí puede verse inmediatamente como un mismo archivo puede ser visto con nombres diferentes por varios usuarios (JOSE/GAMMA y PEPE/ALFA son el mismo archivo).

En LCU la capacidad de acceso a un archivo pertenece al usuario que desea accederlo.

15.4.3. - **Comparación entre LCA y LCU**

Es siempre importante antes de comparar cualquier estructura LCA con la LCU tener en cuenta cómo es la estructura de directorios para acceder a los archivos ya que dependiendo de su complejidad y flexibilidad (pueden existir encadenamientos entre las entradas del DAB, encadenamientos entre las entradas de los archivos de los catálogos de usuarios, etc.) es que pueden sopesarse correctamente sus ventajas y/o desventajas.

Uno de los principales inconvenientes se presenta al querer deletear (borrar, eliminar) un archivo.

En la LCA la baja de un archivo no presenta mayores inconvenientes ya que se elimina la entrada del archivo en el catálogo del usuario Owner y luego se liberan las entradas correspondientes al archivo y a su LCA en el DAB.

En cambio en LCU presenta un inconveniente, ya que no se sabe a priori qué otros usuarios apuntan a ese archivo. Una solución es al eliminar el archivo recorrer **todas** las LCU de los otros usuarios rastreando aquellas cuyo puntero al básico coincida con el archivo eliminado para darlas de baja. Esta tarea se puede ver facilitada si se encadenan las entradas para un mismo archivo que correspondan a diferentes usuarios, pero por otra

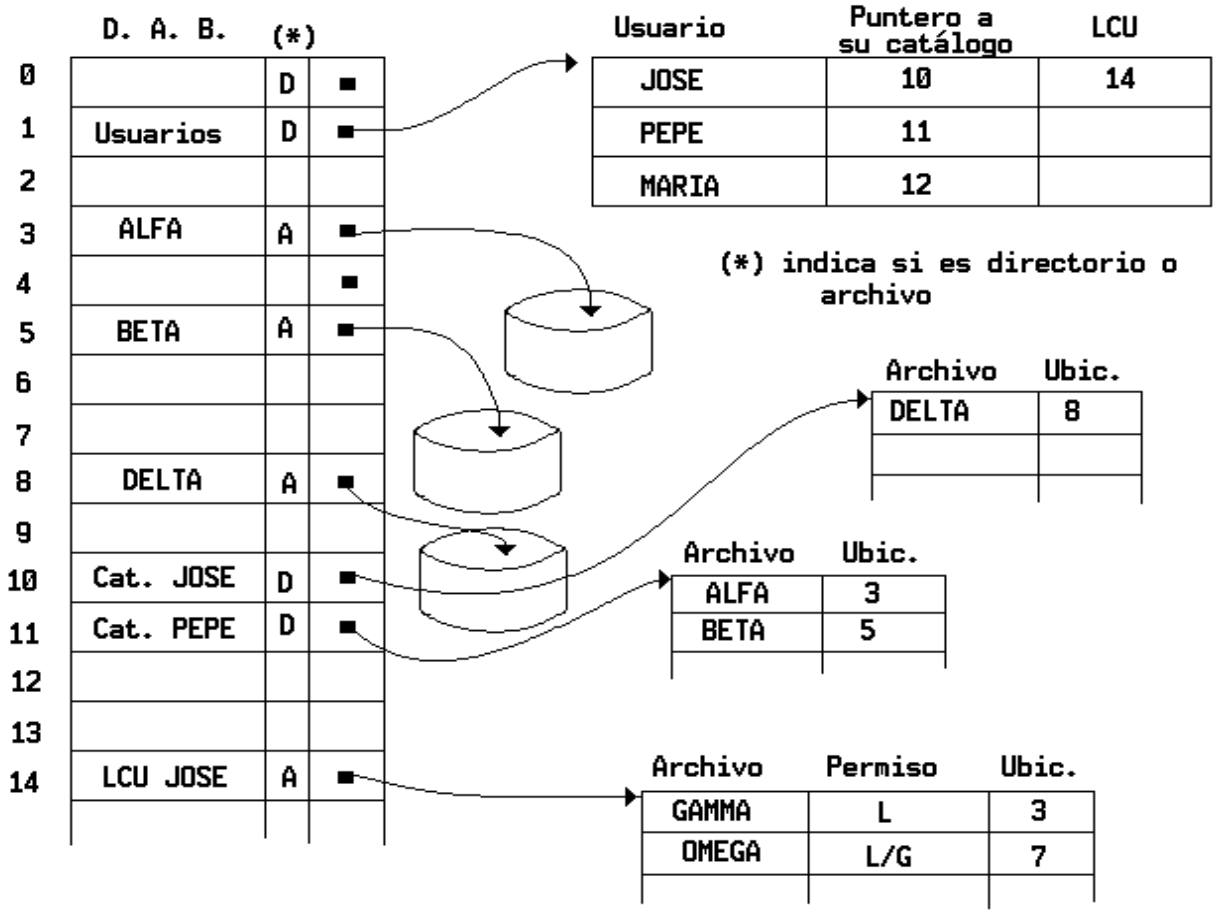


Fig. 15.7. - Sistema de Lista de Control de Usuario (LCU).

parte complica el mecanismo de mantenimiento y actualización de archivos.

La LCU por su naturaleza permite fácilmente la existencia de diferentes nombres para un mismo archivo lo que beneficia al sistema desde un punto de vista de seguridad y confidencialidad, hemos visto que esto no es tan sencillo en el sistema LCA.

No es fácil implementar un permiso genérico (del tipo Todos Leen) en una LCU ya que debería colocarse en la LCU de cada usuario la referencia al archivo deseado.

15.5. - **OPERACION DE OTRAS INSTRUCCIONES DE E/S.**

Hemos visto en detalle en párrafos anteriores las diferentes acciones que desencadena una instrucción de lectura (READ) en el sistema de administración de la información. A continuación detallaremos las acciones que se suceden con otras instrucciones típicas de E/S.

15.5.1. - Instrucción OPEN

Para que un archivo pueda abrirse, éste deberá ser asignado previamente. La asignación puede hacerse dentro del programa con la ejecución de la instrucción de apertura del archivo o separadamente (a nivel etapa).
La asignación propiamente dicha realiza los siguientes pasos :

- generar la entrada para el dispositivo apuntada desde el BCP.
- genera la entrada para el archivo apuntada también desde el BCP y asocia el nombre interno con el nombre externo.
- genera una entrada en la TNA.

En aquellos casos en que el lenguaje no provee una instrucción específica de apertura generalmente la primer instrucción de lectura o grabación del archivo es la que desencadena las acciones de la instrucción OPEN.

15.5.1.1. - **Apertura de archivo con asignación a nivel etapa (OPEN sin ASSIGN)**

Como ya se realizó la asignación, ya existe una entrada en la Tabla de Nombres Activos (TNA), con esta información se procede entonces a buscar el archivo en la Tabla de Archivos Activos (TAA). Si se encuentra, implica que el archivo estaba siendo usado por otro proceso y se controlan los permisos y concurrencia según los campos respectivos en la TAA, y luego se asocia el dispositivo y el archivo al proceso.

Si no se encuentra el archivo en la TAA se deberá generar una entrada en la misma accediendo al Directorio de Archivos Básicos (DAB) en el disco y copiando la entrada correspondiente al archivo en memoria (TAA). Para la búsqueda del DAB es posible que haya que recorrer los volúmenes. Se controlan los permisos en el disco (que también se copian en la TAA) y finalmente se asocia el dispositivo y el archivo al proceso.

Es importante tener en cuenta que si el archivo es de escritura (OUTPUT) y se lo utiliza por primera vez se debe solicitar la asignación del espacio libre al Módulo de Estrategia de Asignación.

15.5.1.2. - **Apertura de archivo con asignación de dispositivos**

Se genera una entrada para el dispositivo y otra entrada para el archivo en el BCP. Como el OPEN debe realizar la asignación, asocia el nombre interno con el nombre externo generando una entrada para ese archivo en la TNA

A partir de este punto la apertura se realiza igual que en el apartado anterior.

15.5.2. - Instrucción CLOSE

Se busca la entrada del archivo en la Tabla de Nombres Activos (TNA). Con el identificador que se extrae de la TNA, se accede a la TAA. Por el Sistema de Archivos Físicos se puede saber si hay algún bloque que deba ser grabado o no (solamente se chequea esto último si el archivo fue utilizado de output o de input-output). Si éste no fue grabado, se debe realizar una E/S física (en este caso un WRITE), se llama al Módulo de Estrategia de Asignación (MEA), luego al Módulo de Estrategia de Periférico y se lanza la E/S física.

Si el archivo se está usando concurrentemente con otros procesos se elimina el proceso de la lista de archivos asociados al proceso en la TAA y se decrementa en uno el campo concurrencia de la misma tabla.

Si el archivo no se usa concurrentemente, es decir el campo concurrencia es igual a 1, se verifica si su entrada en la TAA fue modificada (casos de output, input-output o información de fechas), por lo tanto se debe actualizar esta información en el disco. Con SAB se graba esta información de la TAA al Directorio de Archivos Básicos en el Disco y por último se borra la entrada del archivo de la lista de procesos que lo están usando y se borra la entrada del archivo en la TAA (concurrencia = 0).

15.5.3. - Instrucción DELETE

Para borrar un archivo se debe controlar si el usuario que desea borrar el archivo tiene autorización para hacerlo (VCA - Puede ser el dueño o tener permiso de borrado). Si no está autorizado, se produce un error y no se puede realizar la acción.

Caso contrario, si se trata de un esquema de protección de tipo LCU, se recorren la listas de los usuarios buscando aquellas que apuntan a la entrada del archivo en el Básico y se las elimina. Luego se elimina físicamente el archivo, devolviendo el espacio ocupado por el mismo como espacio disponible. La eliminación física del archivo implica la baja de la entrada del directorio básico que apunta a la dirección física real del archivo en cuestión.

El recorrido según las listas es el siguiente :

LCU

- Por cada usuario del sistema :
  - i - Ingresar en la LCU de los archivos que no le son propios.
  - ii - Eliminar el archivo de la lista
- Para el usuario dueño del archivo :
  - i - Elimina el archivo de la lista de archivos propios.

LCA

- i - Ingresar a la LCA del archivo y la elimina.
- ii - Devuelve también la entrada de la LCA como espacio libre.

Es necesario aclarar que ciertos lenguajes proveen la instrucción DELETE como una de las disponibles. En cambio en otros sistemas la acción de borrar un archivo se especifica por medio de una tarjeta de control. De todas maneras en la mayoría de los sistemas operativos de hoy en día se verifica la concurrencia de la acción de borrado respecto de la utilización del archivo por algún usuario en el momento en que se desea borrar el mismo a efectos de prevenir el fin anormal de una tarea.

15.6. - **ALGUNAS CONSIDERACIONES MAS: Sistemas de Archivo Lógico o Método de Acceso (Dirección lógica).**

El "cálculo de la dirección lógica" depende directamente de la estructura de los registros (formatos), de la organización del archivo y de la forma en que se quiere acceder a la información.

Las formas de llegar a la información, dependiendo de sus estructuras, organización y acceso es lo que conforman los **Métodos de Acceso**.

Luego, los Métodos de Acceso serán distintos (operarán de distinta manera) dependiendo de:
- la forma en que se quiere acceder a la información,
- del formato de los registros y
- la organización del archivo.

Veamos algunos ejemplos.

15.6.1. - **Archivo Secuencial, Formato Fijo.**

En este tipo de archivo los registros tienen igual longitud, como podemos ver en el siguiente grafo de la Fig. 15.8.

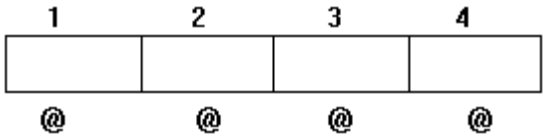


Fig. 15.8.

15.6.1.1. - **Acceso Secuencial**.

En la TAA se mantiene un NBI (Número de Byte Inicial) (dbli) y un NBF (Número de Byte Final) (dblf)
NBI = inicialmente está en cero
NBF = número siguiente al del último registro grabado.
Luego, en este caso el (SAL) Método de Acceso actúa de la siguiente forma:
En lectura, luego de procesar cada registro se cambia
NBI = NBI + LR (siendo LR la longitud del registro)
En escritura, si está permitido escribir luego del último registro (append), se cambia
NBF = NBF + LR

Nótese que el NBI debe estar en la TAA y si el archivo es compartido debe existir además un NBI para cada proceso que lo comparte.

El NBF es un dato que está incluido en la Tabla de Descripción del Archivo, y debe incluirse en la TAA durante la operación de OPEN. Obviamente habrá uno solo.

En el momento de la creación de un archivo NBI y NBF mantienen el mismo valor.

15.6.1.2. - **Acceso Directo**

Aquí se aplica el ejemplo ya visto, donde dado un NR (número de registro) se aplica:
NBI = (NR - 1) * LR

15.6.2. - **Archivo Secuencial, Formato Variable.**

En este tipo de archivos la longitud de cada registro es variable, de la siguiente forma:

Fig. 15.9.

Donde
$L@_i$ = contiene la longitud del registro (sus longitudes son iguales)
$@@_i$ = contiene los datos en sí (sus longitudes son distintas)

15.6.2.1. - **Acceso Secuencial.**

Aquí valen las mismas consideraciones realizadas antes para NBI y NBF.
NBI = inicialmente en cero
luego el próximo registro se encontrará en :
$NBI = NBI + L@_n + @_n$
donde

$L@_n$ = longitud del campo longitud del registro n
$@@_n$ = longitud del dato

Hay que tener en cuenta que es necesario mantener en memoria un buffer de longitud suficiente para albergar el registro más largo posible de este archivo, dato que está en la descripción del mismo archivo.

15.6.2.2. - **Acceso Directo.**

Si se quiere acceder al Registro Número 3, la forma de obtener su dirección es únicamente posible por medio de :

$NBI = L@_1 + @_1 + L@_2 + @_2$

Obviamente acceder a un registro de esta manera es muy ineficiente.

Algunas maneras de mejorar esta situación serían :

1)- Mantener el valor del último NBI de tal manera que si el próximo registro que se busca es superior al anterior buscado, se comienza desde ese.

2)- Mantener el valor de todos los NBI leídos para usos futuros (Note/Point) (con la esperanza de reusarlos)

3)- Mantener en memoria una tabla para todos los NBI (tiene la desventaja de una gran ocupación de espacio en memoria)

Para otras organizaciones será necesario que el Método de Acceso consulte las estructuras propias de esa organización.

Por ejemplo, en un Secuencial Indexado, dada su clave se deberá encontrar en los índices el valor del NBI; en un Random con clave, de deberá calcular a través de la clave y una función (si es provista) el valor del NBI.

A medida que las estructuras son más complejas y se brindan mayores facilidades de búsqueda (a través de relaciones, etc.) dentro del mismo archivo, e inclusive entre distintos archivos (referencias cruzadas, etc.) se encuentra que los Métodos de Acceso se transforman en Sistemas de Gestión de Bases de Datos.

15.6.2.3. - **Sistemas de Archivos Físico (SAF).**

El cálculo de la dirección física se realiza de la siguiente manera:

Número de Bloque Relativo (NBR) = [DBL / Longitud Bloque] y

DF = NBR + Dir. Archivo (o dirección del 1er Bloque)

Además se ubica la posición de la información dentro del Bloque (byte dentro del bloque) como :

Posición Relativa Registro (PRR) = Resto [DBL / Longitud Bloque]

Si DF es una dirección cuyo contenido ya se encuentra dentro del buffer en memoria esto significa que **no** es necesario realizar una operación de E/S física, sólo es necesario trasladar la información al área del proceso (o pasar la dirección de la información), en caso contrario se sigue adelante con el resto de las funciones de la Administración de Archivos y las operaciones de E/S físicas.

Nótese aquí que si se estuviera trabajando con un Sistema de Administración de Memoria Paginada por Demanda, y el buffer no se encontrase en memoria se produciría una Interrupción por Falta de Página.

Pueden existir situaciones especiales como el caso de un registro lógico que supere el tamaño de los registros físicos, como por ejemplo :

CdB: Cabecera de bloque (usualmente 4 bytes de longitud, indica la longitud del bloque)

Fig. 15.10.

Aquí habría que determinar cuantos bloques es necesario traer a memoria, que se calcula como:

Cantidad de bloques = [Long. Lógica / Long. Bloque] + 1

si PRR = 0, o sea el registro comienza al comienzo del bloque y

Cantidad de bloques = [Long. Lógica / Long. Bloque] + 2

si PRR ¹ 0, o sea que el registro comienza en el interior de un bloque.

Fig. 15.11.

Una vez que la información está en el buffer se trasladan @ bytes al área del usuario.

15.7. – **FILE SYSTEMS DE EJEMPLO**

15.7.1. - **File System en UNIX**

Se utiliza como base el UNIX 4.3BSD (Berkeley Software Distribution) en máquinas VAX (1987).

15.7.1.1. - **Manejo de Archivos**

Para algunos sistemas operativos los archivos son conocidos como estructuras, en UNIX sin embargo un archivo es una secuencia de bytes, el kernel no conoce estructuras.

Los archivos se organizan en estructuras llamadas directorios. Los directorios son archivos que tienen información para encontrar a otros archivos.

Un path name a un archivo es un string que indica el camino a través de las estructuras de directorios hacia un archivo. Se lo denota separando cada nivel con una "/".

Pathname absolutos : empiezan en la raíz
Pathname relativos : empiezan en el directorio actual

Un archivo puede tener distintos nombres. Estos se conocen como links.

Hay dos tipos de links, los soft links tienen el path absoluto al archivo y pueden apuntar a directorios y a archivos en otros file systems.

En cambio los hard links no pueden apuntar a directorios ni a files en otros file systems.

"." es un hard link al directorio actual
".." es un hard link al directorio padre del directorio actual

Los dispositivos de hardware tienen nombres en el file system.

Estos archivos especiales de dispositivo o archivos especiales son conocidos por el kernel como interfases de dispositivos y son accedidos por el usuario con las mismas llamadas al sistema que para acceder a otros archivos.

/ Raíz	vmunix	Imagen binaria de booteo del UNIX	
	dev	Archivos especiales de dispositivos. Por ej. /dev/console, /dev/lp0, /dev/mt0	
	bin	Archivos esenciales del UNIX	
	Lib	Archivos de bibliotecas de C, Pascal, subrutinas FORTRAN	
	Usr	Bin	Programas de aplicación del sistema (editores de texto)
		Local	Bin: programas del sistema escritos localmente
	user	Para usuarios	
	etc	Archivos administrativos, por ej. Password	
	tmp	Temporarios	
EJEMPLO DE CONTENIDOS DE LOS DIRECTORIOS EN UNIX 4.3 BSD			

Las llamadas básicas del sistema para manipular archivos son :

- *Creat* : dado un path crea un archivo vacío o trunca uno ya existente.
- *Open* : abre un archivo. Dado un path y un modo (read, write o r/w) devuelve un entero pequeño que se denomina file descriptor. Este descriptor se pasa a la llamada read o write junto con la dirección del buffer y la cantidad de bytes a transferir.
- *Close* : se cierra un archivo cuando se pasa a esta llamada el file descriptor de ese archivo.
- *Trunc* : reduce la longitud de un archivo a cero
- *Lseek* : permite resetear explícitamente la posición dentro del archivo
- *Dup* o *dup2* : crea copias del file descriptor
- *Fcntl* : hace lo mismo que dup pero además puede ver o setear algunos parámetros de un archivo abierto, por ejemplo puede hacer que cada write al archivo realice un append.
- *Ioctl* : se usa para manejar parámetros de los dispositivos.
- *Stat* : devuelve información sobre el archivo por ejemplo tamaño, dueño, permisos, etc.
- *Rename* : cambia el nombre del archivo
- *Chmod* : cambia los permisos del archivo
- *Chown* : cambia el owner del archivo
- *Link* : crea un hard link para un archivo existente con un nuevo nombre
- *Unlink* : remueve un link. Si es el último el archivo se borra
- *Symlink* : crea un link simbólico
- *mkdir* : crea un directorio

- rmdir : borra un directorio
- *cd* : cambia de directorio

Un file descriptor es un índice a una tabla de archivos abiertos por este proceso. Van de 0 a 6 o 7 dependiendo de la cantidad de archivos abiertos que tenga el proceso.

Cada read o write actualiza el desplazamiento dentro del archivo que se encuentra asociado con la entrada en la tabla de archivos y se usa para determinar la posición en el archivo para el próximo read o write.

Las instrucciones para operar sobre directorios tienen que ser distintas de las anteriormente mencionadas, ya que la estructura interna de un directorio debe ser preservada.

Estas son : opendir, readdir, closedir, etc.

15.7.1.2. - **FILE SYSTEM**

UNIX soporta principalmente dos tipos de objetos: Archivos y Directorios. Los directorios son archivos con un formato especial.

15.7.1.2.1. - **Bloques y Fragmentos**

La mayoría del file system se mantiene en bloques de datos. Un tamaño de bloque mayor a 512 bytes es bueno por cuestiones de velocidad pero como en UNIX usualmente existen archivos de tamaño pequeño los bloques muy grandes causarían demasiada fragmentación interna.

Una solución consiste en utilizar 2 tamaños de bloques: Todos los bloques de un archivo con de un tamaño suficientemente grande, por ejemplo 8K, y el último bloque es un múltiplo de un pequeño fragmento, por ejemplo 1024 bytes.

Ejemplo: Un archivo de 18.000 bytes tendría 2 bloques de 8K y un fragmento de 2K.

Los tamaños de los bloques y fragmentos se especifican en la creación del file system.

Detalles de la implementación fuerzan a una relación bloque:fragmento de 8:1 y bloques con un mínimo de 4K, por ejemplo relaciones del estilo 4096:512 o 8192:1024 son válidas.

A medida que se generan un fragmento se copia en disco y si se genera otro fragmento entonces se copia y se junta el primer fragmento con el segundo, por lo tanto pueden llegarse a realizar 7 copias antes de obtener un bloque. Para evitar esto se proveen mecanismos para que los programas descubran el tamaño del bloque y evitar la recopia de fragmentos.

15.7.1.2. - **Inodos**

Un archivo se representa a través de un inodo (o i-nodo). Un inodo es un registro que almacena la mayoría de la información específica de un archivo en disco.

Inodo deriva de las palabras “nodo índice”, antiguamente se escribía como i-nodo pero con el uso se reemplazó por inodo.

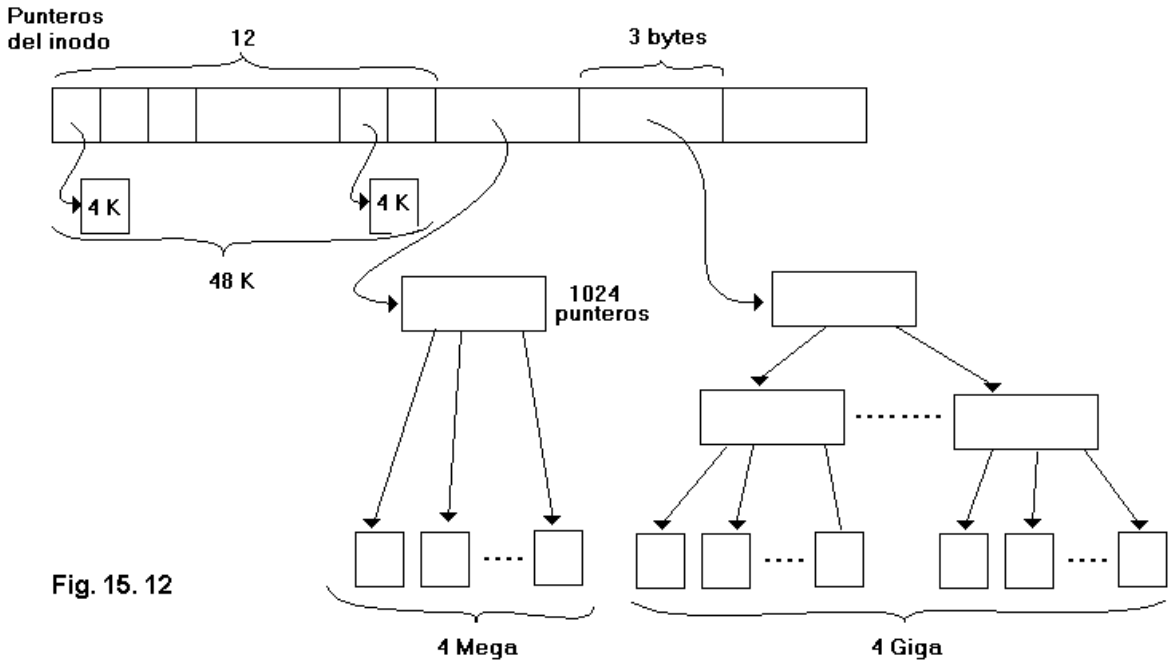


Fig. 15. 12

Un inodo contiene :

- identificadores del usuario y del grupo,

- hora de la última modificación y acceso al archivo
- un contador de hard links que apuntan al archivo
- tipo de archivo (plano, directorio, link simbólico, dispositivo de caracteres, disposit6ivoi de bloques o socket)

El inodo tiene 15 punteros a bloques de disco que contienen los datos. Los primeros 12 apuntan directamente a bloques. Si el archivo no es mayor a 12 bloques se puede acceder directamente por que el inodo se mantiene en memoria mientras el archivo está abierto.

Los otros 3 punteros usan direccionamiento indirecto. Los bloques apuntados así son de tamaño de bloque no de fragmento que se aplica solo a datos.

El primero de los 3 punteros es una indirección directa, el 2do es una doble indirección y el 3ero es una triple indirección aunque en general no llega a ser necesaria esta última ya que con bloques de 4K alcanza con doble indirección para un archivo de hasta 2^{32} bytes (4 Gigabytes).

En UNIX System V los bloques son de 1K y por lo tanto el máximo con doble indirección llega a 65 MB y con triple indirección llega a 16 GB.

El desplazamiento dentro de un archivo se maneja en memoria con una palabra de 32 bits. Los archivos, sin embargo, no pueden ser mayores a 2^{32} bytes.

15.7.1.3. - **DIRECTORIOS**

A este nivel no hay diferencia entre archivos planos y directorios. El contenido de los directorios se mantiene en bloques y se los representa con un inodo como a un archivo.

La única diferencia es el campo tipo en el inodo.

Los nombres de archivos son de longitud variables hasta 255 caracteres y por lo tanto las entradas en los directorios son de longitud variable. Cada entrada tiene primero su longitud, luego el nombre del archivo y el número de inodo.

Los primeros 2 nombres en todo directorio son “.”y “..” Las búsquedas en el directorio se hacen linealmente.

El usuario se refiere a un archivo a través de su path-name, sin embargo el file system usa el inodo y su definición del archivo, luego el kernel tiene que mapear el pathname que el usuario dio con el inodo. Para ello se usan los directorios.

Se parte desde el primer calificador del path (si es / es el raíz) y se obtiene su inodo. Se chequea el tipo de inodo y los permisos y se continúa con cada uno de los calificadores siguientes hasta obtener el inodo del archivo.

Los hard links se tratan como cualquier otra entrada del directorio.

Con los links simbólicos se comienza la búsqueda con el pathname que contenga el link y se previenen loops infinitos poniendo un límite de 8 links simbólicos.

Los archivos que son archivos de discos, por ejemplo los dispositivos, no tienen bloques de datos alocados en el disco. El kernel los reconoce por el tipo en el inodo e invoca a los drivers apropiados para manejar su E/S.

Una vez que se ubicó al inodo (por ejemplo a través de un open), se aloca una estructura de archivo que apunta al inodo. El descriptor de archivo que se entrega al usuario apunta a esta estructura de archivo.

15.7.1.4. - **Mapeando un descriptor de archivo con un inodo**

Las llamadas al sistema que referencian a archivos abiertos indican el archivo pasando un descriptor de archivo como argumento.

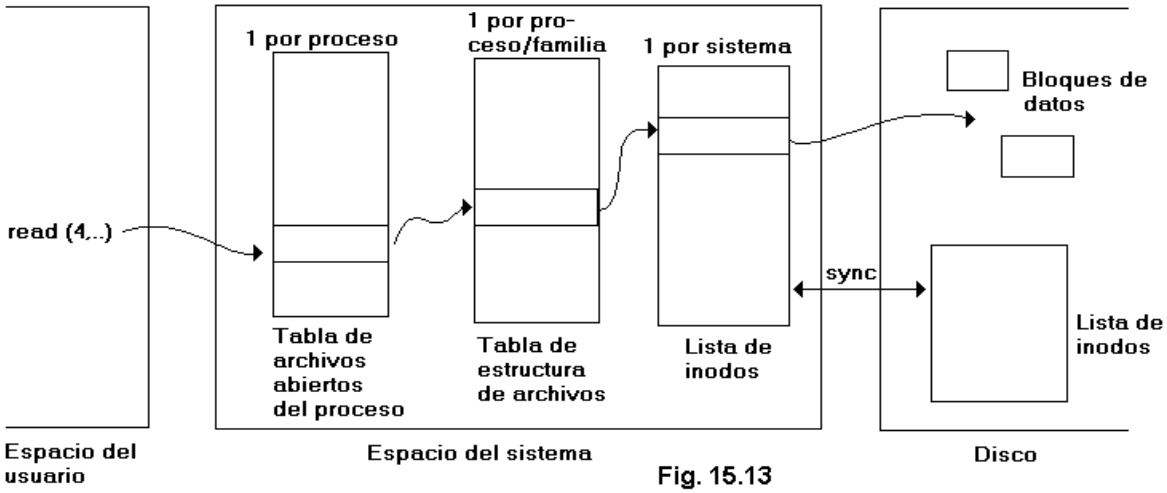


Fig. 15.13

El descriptor es usado por el kernel; para indexar una tabla de archivos abiertos por ese proceso. Cada entrada en la tabla tiene un puntero a una estructura de archivo. Esta estructura apunta al inodo.

Diferentes nombres de archivos pueden estar asociados con un mismo inodo pero un inodo que está activo está asociado exactamente con un solo archivo y cada archivo es controlado exactamente por un solo inodo.

El kernel mantiene un desplazamiento dentro del archivo que se actualiza con cada read o write.

Ya que más de un proceso puede estar usando concurrentemente un mismo archivo no es práctico llevar el offset en el inodo y por ende se lo mantiene en la estructura de archivo.

Las estructuras de archivos se heredan cuando un proceso padre crea un hijo (instrucción fork) y por lo tanto distintos procesos pueden tener el mismo offset dentro del archivo.

La estructura de inodos apuntada desde la estructura de archivos es una copia en memoria de la lista de inodos en el disco. Esta estructura se mantiene en una tabla de longitud fija.

La tabla en memoria tiene algunos campos extra como ser un contador de cuántas estructuras de archivos la apuntan y la estructura de archivos tiene un campo similar sobre cuántos descriptores de archivos la apuntan.

15.7.1.5. - **Estructura de Discos**

El usuario usualmente conoce solo un file system pero un file system lógico puede contener varios file systems físicos.

Usualmente se particionan los discos físicos en múltiples dispositivos lógicos, Cada dispositivo lógico define un file system.

Las ventajas de este enfoque son :

- distintos file systems pueden soportar diferentes usos
- es más confiable ya que una falla de software daña solo a ese file system
- cada file system puede tener diferentes tamaños de bloques/fragmentos
- evita que un solo usuario consuma el espacio total de los dispositivos físicos ya que está limitado al tamaño del file system

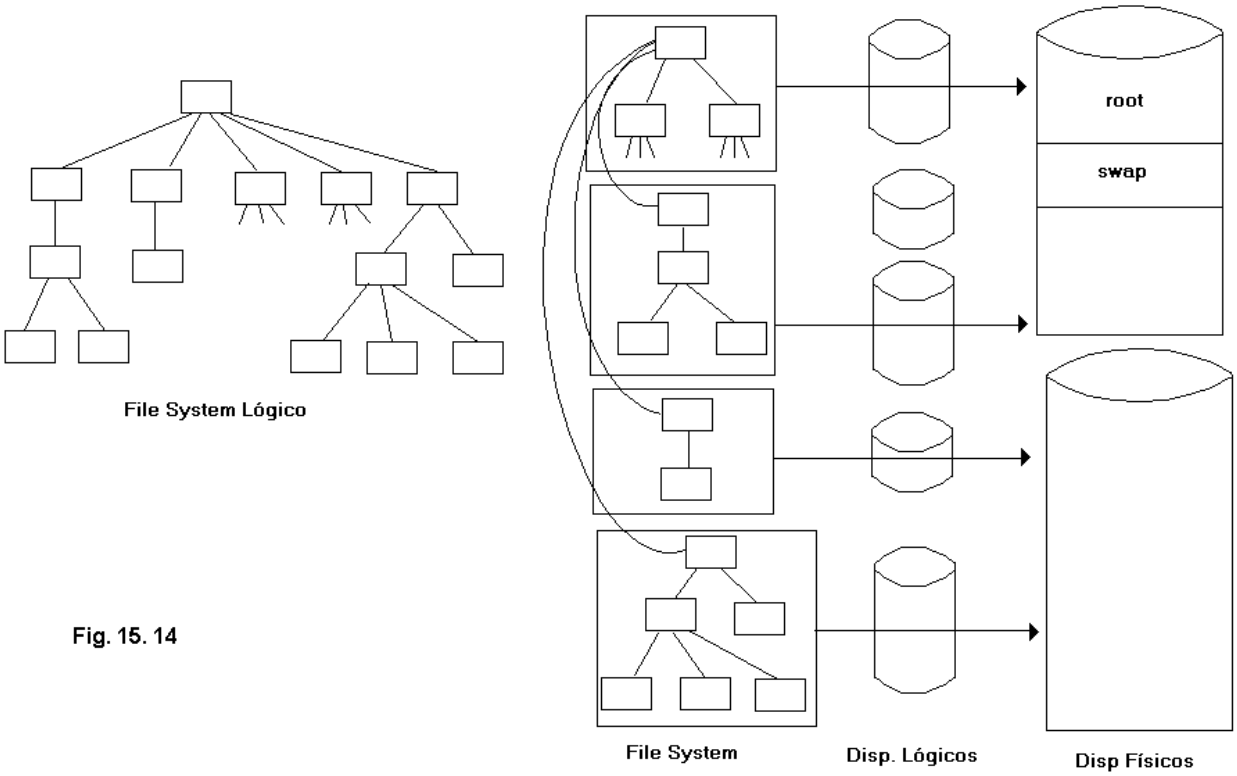


Fig. 15. 14

El file system raíz siempre está disponible y los otros pueden montarse, es decir se integran en la jerarquía del raíz.

Hay un bit en el inodo que indica si sobre él se ha montado un file system Una referencia a este archivo provoca que la tabla de montaje sea recorrida para ubicar el número de dispositivo montado.

El número de dispositivo se usa para encontrar el inodo del directorio raíz del file system montado y ese inodo es el que se utiliza.

De forma similar si un pathname es ".." y el directorio a buscar es el raíz de un file system montado, se recorre la tabla de montaje para encontrar el inodo sobre el cual se montó y se utiliza ese.

15.7.2. - **Sistema de Archivos de DOS**

15.7.2.1. - **Un poco de historia**

En 1977 Microsoft crea un sistema de archivos y lo llama FAT (File Allocation Table). Luego adaptaría la FAT para el sistema operativo DOS (Disk Operating System).

Originalmente el DOS fue pensado para ser usado desde un disco flexible, es por eso que la primer versión de FAT (FAT-12) puede almacenar como máximo 8Mb de información.

Este error de diseño es solo comparable con el error de diseño respecto al direccionamiento de la memoria de los procesadores de la línea 80x86 (“640k deben ser suficientes”).

Viendo que 8Mb era poco Microsoft decidió dar un salto en tecnología y diseñar FAT-16, ahora la FAT podría almacenar los inalcanzables 32Mb!

En 1987 nuevamente los 32Mb ya no igualaban el tamaño de los discos y entonces se decidió atacar el problema de manera definitiva.

Viendo que 32Mb era poco se diseñó FAT-32, ahora la FAT podía almacenar 2Gb.

En 1995 ya existían algunos discos mayores a 2Gb.

Entonces era hora de aumentar nuevamente el tamaño de la FAT.

Viendo que 2Gb era nuevamente poco se diseñó VFAT (o FAT-32 extendido), ahora la FAT podía almacenar 2Tb.

Resumiendo:

Versión DOS	Nombre de FAT	Cantidad de Clusters	Cantidad Sectores por Cluster	Tamaño máximo de un archivo y de una partición
DOS 2.0	FAT-12	4Kb	4	8Mb
DOS 3.3	FAT-16	16Kb	4	32Mb
DOS 4	FAT-32	65Kb	64	2Gb
DOS 7(Win95)	VFAT (FAT-32 ext.)	4Gb	64	2Tb

En la actualidad el sistema de archivos de Windows 2000 soporta 64Tb, por ahora parece ser suficiente.

15.7.2.2. - **Introducción**

A continuación se explicara la estructura de las FAT-12 a FAT-32. Dado que cada nueva FAT es compatible con la anterior se tomara como base la FAT-12 agregando la información necesaria para extender los conceptos a FAT-16 o 32.

15.7.2.2.1. - **Algunas definiciones**

El ***sector*** es la unidad mínima de almacenamiento de los discos. Un sector tiene 512 bytes.

Un ***cluster*** es un conjunto contiguo de sectores que se agrupan para manipular mayor cantidad de información al mismo tiempo. Cada cluster tiene al menos 1 sector.

Cada archivo puede ocupar 1 o mas clusters, no necesariamente contiguos.

FAT es una sigla en ingles que significa “Tabla de ubicación de archivos”.

15.7.2.2.2. - **La estructura de la FAT en DOS**

Si se tiene una FAT-16 se dispondrá de 65k clusters disponibles para almacenar 65k archivos o directorios en un principio.

Sin embargo el DOS utilizara algunos de ellos para almacenar información del sistema operativo. Ademas de estos clusters hay otros reservados para otros usos que no pueden ser usados por archivos del usuario.

Cluster Numero	Contenido
Cluster 0	Reservado para DOS
Cluster 1	Reservado para DOS
Cluster 2	2 (usado para almacenar un archivo chico)
Cluster 3	4 (usado para almacenar datos, extendido al cluster 4)
Cluster 4	5 (usado para almacenar datos, extendido al cluster 5)
Cluster 5	7 (usado para almacenar datos, extendido al cluster 7)
Cluster 6	0 (vacío, disponible para uso)
Cluster 7	FFFh (usado para almacenar datos, el ultimo “eslabón” de la cadena)
Cluster 8	0 (vacío, disponible para uso)
Cluster 65524	0 (vacío, disponible para uso)
Cluster 65525	0 (vacío, disponible para uso)
Cluster 65526	0 (vacío, disponible para uso)

15.7.2.3. - Capas lógicas en un disco FAT

Los sectores de un disco FAT están divididos en estos grupos, están listados en orden creciente al numero de sector.

1. Sectores reservados
2. FAT's
3. Directorio Raíz
4. Area de datos y Sectores ocultos

15.7.2.3.1. - **Sectores reservados**

En el sector lógico 0 se encuentra el bootsector [sector de arranque]
Contiene 512 bytes con la siguiente información.

Byte Nro.	Contenido	Descripción
0-2	Primera instrucción de la rutina de arranque	
3-10	Nombre OEM	Normalmente la versión de DOS, opcional.
11-12	Cantidad de bytes por sector	Múltiplos de 512
13	Cantidad de sectores por cluster	
14-15	Cantidad de sectores reservados	
16	Cantidad de copias de la FAT	Usualmente 2
17-18	Cantidad de entradas en el directorio raíz	
19-20	Cantidad total de sectores	
21	Descriptor del dispositivo	Disco 3½ / Disco 5¼ / Disco Rígido / Etc.
22-23	Cantidad de sectores en cada copia de la FAT	
24-25	Cantidad de sectores por pista	
26-27	Cantidad de lados	
28-29	Cantidad de sectores ocultos	
30-509	Rutina de arranque e información de la partición	
510	55(en hexadecimal)	
511	AA(en hexadecimal)	

15.7.2.3.2. - **FAT's**

En los sectores siguientes inmediatos se encuentran las copias de la FAT.
La Segunda copia de la FAT solo se utiliza para restaurar la 1ra en caso de necesidad.
Una FAT se almacena como una secuencia de registros de igual tamaño.
El tamaño del registro lo determina la versión de la FAT.
Para la FAT-12 el registro es de 12 bits. Para la FAT-16 en cambio es de 16 bits.

Cada registro codifica la siguiente información:

12-bit	16-bit	Significado
000	0000	Cluster disponible
001	0001	Entrada invalida
002-FEF	0002-FFEF	Cluster asignado, el contenido es el nro. del próximo cluster del mismo archivo en el mismo directorio
FF0-FF6	FFF0-FFF6	Reservado
FF7	FFF7	El cluster contiene sectores defectuosos
FF8-FFF	FFF8-FFFF	Cluster final de un archivo.

Dado que un archivo es una **lista encadenada de clusters**, el tamaño del archivo esta limitado por el tamaño de la partición FAT.

15.7.2.3.3. - **Directorio Raíz**

El directorio raíz contiene un registro de 32 bytes por cada archivo que esta en el.
Tiene una entrada mas para la etiqueta del volumen.

15.7.2.3.4. - **Area de datos y sectores ocultos**

La cantidad total de sectores codificadas en los bytes 19 y 20 del sector de arranque incluyen todas las áreas salvo la de sectores ocultos.

La cantidad total de sectores se calcula sumando a esa cantidad la cantidad de sectores ocultos que figuran en los bytes 28 y 29.

No se conoce un uso especifico para los sectores ocultos. Podrían usarse para “camuflar” otra partición o también para almacenar sectores con formatos especiales. Usualmente son pocos.

15.7.2.4. - **Estructura de directorios**

Cada entrada de directorio tiene la misma estructura:

Byte Nro.	**Contenido**
0-7	Nombre del archivo u 8 primeras letras del nombre del volumen
8-10	Extensión del archivo o 3 ultima letras del volumen
11	Byte de atributo*****
12-21	No usado
22-23	Hora (bit 15-11 horas [0-23]; bit 10-5 minutos [0-59]; bit 4-0 “doble” segundos [0-29])
24-25	Fecha
26-27	Dirección del primer cluster
28-31	Cantidad de bytes del archivo. (Si es un subdirectorio o etiqueta de volumen vale 0)

*Significado del byte de atributo

Bit Nro.	Significado (bit = 1 = ON)	Bit No	Significado (bit = 1 = ON)
0	Solo lectura	4	La entrada representa un subdirectorio
1	Archivo oculto	5	El archivo fue modificado desde su ultimo backup
2	Archivo de sistema	6	No usado
3	La entrada representa una etiqueta de vol.	7	No usado

Si una entrada representa un directorio se setea en bit 4 y el resto no es usado.

Si una entrada representa una etiqueta de volumen se setea el bit 3 y el resto no es usado.

DOS no es case-sensitive, es decir no distingue las mayúsculas de las minúsculas para los nombres de archivo.

Campo fecha:
Bits 15-9: Años desde 1980 (0 a 127)
Bits 8-5: Mes (1 = Enero…12 = Diciembre)
Bits 4-0: Día (1 a 31)

Por lo tanto el “mundo DOS” empezó el 1 de enero de 1980 (00:00:00) y terminara el 31 de diciembre de 2107 (23:59:58).

El 1er carácter de una entrada de directorio tiene un significado especial para FAT:
Si es 00h significa que es el ultimo registro.
Si es 05h significa que la primer letra del nombre es el valor ASCII de E5h.
Si en cambio el valor es E5 significa que el directorio esta borrado.

Cuando un directorio es borrado su primer byte es reemplazado por el valor hexadecimal E5 para marcar que el directorio ha sido borrado.

El resto de la cadena no es modificada.

Esto es bueno por un lado porque:

- Permite la recuperación de archivos borrados.
- La operación de borrado es relativamente rápida.

Sin embargo presenta los siguientes problemas:

- El borrado genera fragmentación de los sectores.
- Es posible obtener nuevamente la información borrada lo que representa una falla de seguridad.
- Como lo único que se borra realmente del archivo es la 1er letra (esta fue reemplazada por E5) es necesario “conjeturar” este dato al momento de la recuperación del archivo.

15.7.3. - **HPFS**

15.7.3.1. - **Introducción**

Este sistema de archivos fue utilizado por primera vez en el sistema operativo OS/2 v1.2 para solucionar todos los problemas que la FAT no podía resolver. Las versiones 1.0 y 1.1 utilizaban FAT.

HPFS es una sigla en ingles (High Performance File System) que significa “Sistema de archivos de alta performance”.

15.7.3.2. - **Estructura de un volumen HPFS**

Un volumen HPFS utiliza un sector de 512 bytes y tiene un tamaño máximo de 2199GB.
Los discos removibles pueden usar HPFS, pero usualmente usan FAT pues:

- Es mas fácil transportar los archivos entre FAT, HPFS y otros sistemas mas fácilmente.
- Dado el tamaño de los discos removibles no se aprecian las mejoras de HPFS.

Notar que una de las estrategias de HPFS es mantener los datos tan contiguos como sean posibles, sin desperdiciar espacio para lograr eso.

15.7.3.2.1 - Estructuras fijas de un volumen HPFS:

Sector Nro.	Nombre	Descripción
0-15	BootBlock	32bit (Vol. Id) y programa de arranque.
16	SuperBlock	Punteros a: espacio libre, lista de sectores defect., directorio raíz y directorios. Fecha de ultima chequeo del volumen (CHKDSK/F)
17	SpareBlock	Punteros y Flags

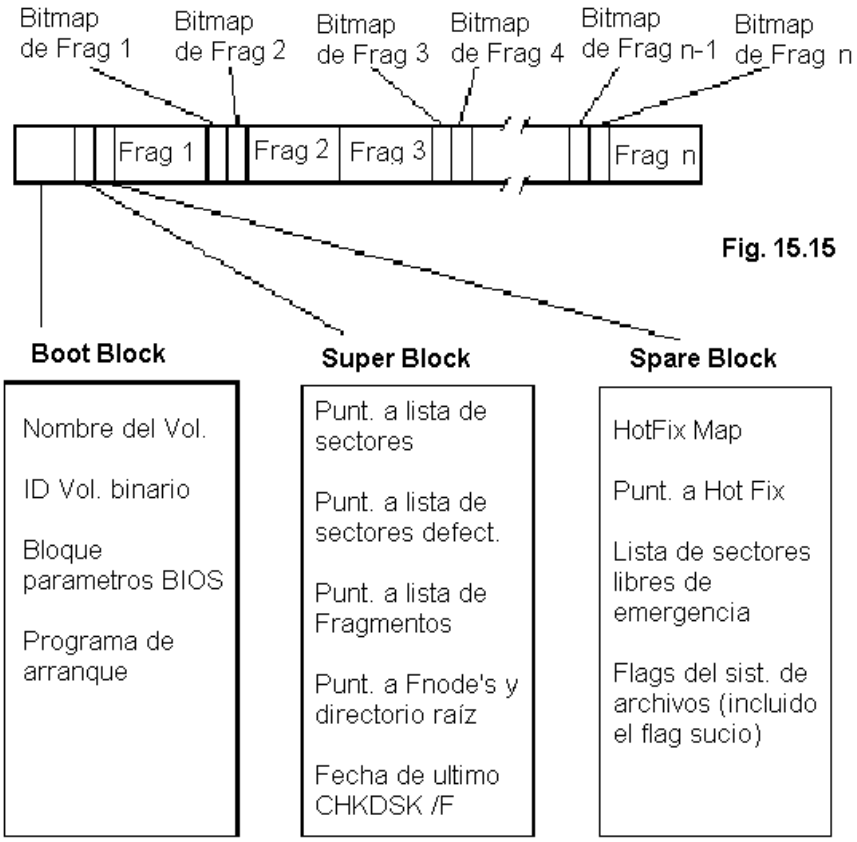


Fig. 15.15

El resto del disco se divide en fragmentos de 8mb. Ver *Figura 15.15*.

Cada fragmento tiene su propio mapa de sectores libres. Como cada bit representa un sector, a este mapa se lo denomina bitmap (mapa de bits) de sectores libres donde cada bit es interpretado así:

- Si el bit = 0: el sector esta ocupado
- Si el bit = 1: el sector esta libre

Este mapa esta situado al principio o al final de cada fragmento, esto permite reservar 16mb de espacio libre contiguo para un mismo archivo.

Hay un fragmento especial localizado en el centro del disco que se denomina bloque de directorio y tiene un tratamiento especial.

15.7.3.3. - **Archivos y Carpetas (Fnode's)**

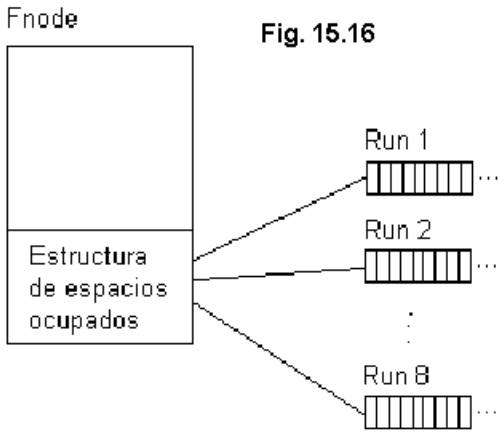


Fig. 15.16

Cada archivo o carpeta esta asociado a una estructura que se denomina Fnode. Ver *Figura 15.16*.

Cada Fnode ocupa un sector y contiene los siguientes datos del archivo o carpeta:

- Información histórica
- Información de control
- Los atributos
- La lista de control de acceso
- La longitud
- Los primeros 15 caracteres del comienzo del nombre
- La estructura de espacios ocupados

HPFS ve al archivo como una colección de runs[1], donde cada run es una secuencia de sectores contiguos.

Cada estructura de espacios ocupados puede contener hasta ocho punteros a runs distintos de manera que el espacio total de almacenamiento que puede soportar un fnode es de 16Mb.

Si un archivo ocupa mas de 16mb no podrá ser almacenado de manera contigua.

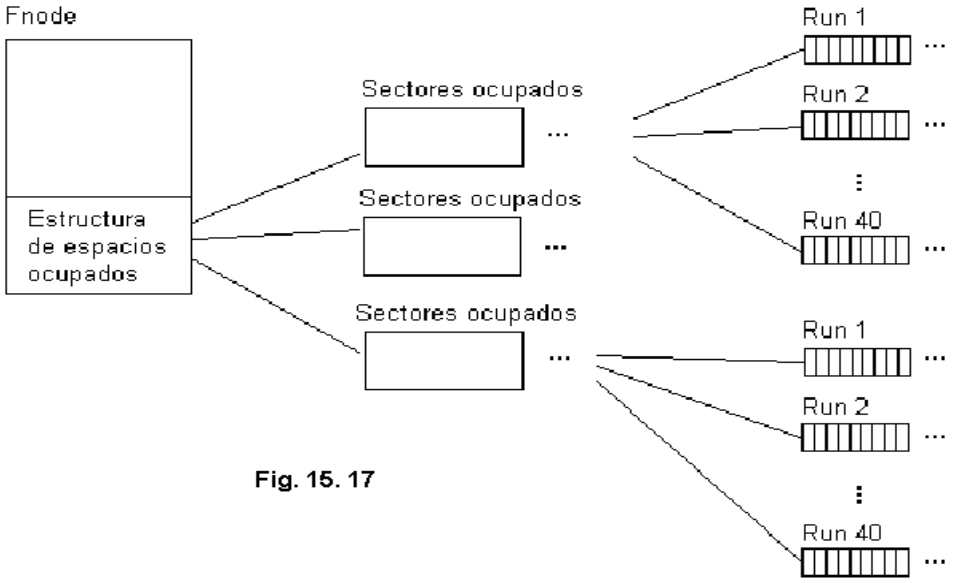


Fig. 15. 17

15.7.3.1. – Estructura.

La estructura del run es la siguiente:

- 32bit: dirección de comienzo del 1er sector
- 32bit: cantidad de sectores

Desde el punto de vista de una aplicación un archivo se ve como una secuencia de bytes.

Para los archivos mayores a 16mb o que se encuentran muy fragmentados HPFS utiliza estructura de espacios ocupados. Esta estructura es básicamente un árbol-b (B-Tree), Ver *15.16*. Mas adelante explicaremos de qué se trata esta estructura de datos.

15.7.3.3.1. - Estructura de un árbol-b en general:

Un árbol-b es un árbol n-ario donde cada nodo puede contener mas de un elemento, en una lista ordenada.

Esta agrupación se utiliza para minimizar la modificación de la estructura en las inserciones y acelerar los accesos a los datos.

Si la agrupación es mínima (un elemento), el árbol-b es un árbol n-ario, y no se aprovecha la particularidad de esta estructura.

[1] El run de HPFS es conceptualmente similar al cluster de DOS: una secuencia de sectores contiguos.

Si la agrupación es máxima, (n elementos), el árbol-b pasa a ser una secuencia y por lo tanto se pierden las propiedades de árbol, o sea es como trabajar con una secuencia.

A la cantidad de elementos que se pueden almacenar como máximo en un nodo se lo denomina orden. *Orden* del árbol-b.

Para un Sistema de archivos, lo mas apropiado es que el *orden* del árbol-b sea múltiplo del tamaño del sector, o del run, en el caso de HPFS.

En cierta forma un árbol-b es una extensión de los árboles 2-3, los cuales se diferencian de los árboles comunes en que no tienen una cantidad de hijos por nodo estricta. Esta pequeña libertad tiene grandes implicancias al utilizar la estructura sobre un disco. Dado que muchas hojas se encuentran parcialmente llenas, es altamente probable que al insertar un nuevo elemento no haga falta generar un nuevo nodo, y aún si esto último fuese necesario, es probable que no haya que generar un nuevo nodo como padre de las hojas, etc. El resultado neto es la *minimización de los nodos modificados* al efectuar una inserción, a costa de una mayor ineficiencia en el aprovechamiento del espacio.

Los árboles B se diferencian de los 2-3 solamente en la cantidad de hijos que pueden tener. Mientras que un árbol 2-3 puede tener 2 o 3 hijos (si no es hoja, claro está) un árbol B de orden n puede tener entre n/2 y n hijos. Esto permite *adaptar el orden del árbol según el tamaño de nodo que se desee obtener*, característica importante si se desea utilizar exactamente un sector por nodo, como se comentó anteriormente.

Otra diferencia importante de un árbol-b respecto a un árbol n-ario es que los datos se almacenan en las hojas mientras que en los nodos internos solo se almacenan referencias a los otros nodos.
En los árboles n-arios cada nodo tiene un dato, y todos los nodos (hojas y nodos internos) son iguales.

Esta estructura garantiza las siguientes características:

- Inserción en $O(\log_{orden} n)$
- Búsqueda en $O(\log_{orden} n)$
- No depende de la “buena” distribución de los datos para balancearse.
- No requiere estar totalmente en memoria para garantizar estos ordenes (una cache acompaña casi siempre a esta estructura pues no siempre es conveniente ni se puede cargar todo el árbol al mismo tiempo).

15.7.3.4. - **Directorios**

Los directorios también están representados por los Fnode’s y en el SuperBlock se encuentra en puntero al directorio raíz.

Los directorios están almacenados en bloques de 2k (cuatro sectores consecutivos) y pueden crecer mientras haya espacio disponible.

Normalmente se almacenan en el fragmento de directorio, que esta cerca del centro del disco para accederlo de forma mas eficiente.

Si este fragmento no es suficiente se utiliza el primer fragmento libre disponible.

Cada bloque de directorio tiene una o mas entradas de directorio, y cada entrada de directorio tiene los siguientes campos:

- Sello temporal
- Puntero a Fnode
- Contador de uso (para los programas de mantenimiento y cache)
- Nombre
- Puntero a un árbol-b

Cada bloque comienza con un numero que indica el tamaño de cada entrada, de manera que el tamaño sea aprovechado sin perder la performance.

En promedio los nombres de directorio contienen 13 letras y alrededor de 40 entradas.

Cada bloque esta ordenado por orden alfabético y la ultima entrada se usa para marcar que no hay mas entradas.

Si el bloque no es suficiente para almacenar un directorio, se utiliza un árbol-b para almacenar la información restante. La cantidad de entradas variable de un bloque hace que el árbol-b sea una estructura apropiada para almacenar este tipo de información.

Para ilustrar la performance que se logra con este tipo de estructura tomaremos como ejemplo el siguiente caso:

> Supongamos que tenemos 40 entradas por directorio, un árbol-b de dos niveles podrá almacenar 1640 entradas y uno de 3 niveles, 65.640.

Un archivo particular podrá ser encontrado en un directorio de 65.640 entradas como mucho con 3 accesos al disco.

Bajo las mismas hipótesis, en una FAT de DOS este mismo acceso provocara 4000 accesos al disco en el peor caso.

Uno de los problemas que presenta esta estructura es cuando se necesita agrandar el tamaño de una entrada, por ejemplo, cuando se hace un rename[2] de un archivo.

HPFS soluciona esto manteniendo un pequeño pool de bloque libres en el directorio de emergencia. (El puntero a este bloque se encuentra en el SpareBlock[3]).

15.7.3.5. - **Manejo de errores**

El mecanismo principal para el manejo de errores se llama hotfix (arreglo rápido). El puntero al bloque hotfix se encuentra en el SpareBlock.

Cuando se detecta un sector defectuoso, se toma un sector del pool de sectores del hotfix para remplazar el sector dañado.

El bloque hotfix contiene una secuencia de doble words (32bit x 2) que representan:

- 32bit: Nro. de sector defectuoso
- 32bit: Nro. se sector que lo reemplaza dentro el hotfix

La cache del sistema de archivos amortigua toda la posible perdida de performance que puede causar este acceso indirecto a estos sectores.

Una de las tareas del CHKDSK[4] es vaciar el bloque hotfix.

Por cada reemplazo que existe en el hotfix, CHKDSK, buscara un sector libre (y bueno) dentro de los fragmentos libres regulares donde recolocar el sector originalmente dañado, actualizando los mapas de archivos que sean necesarios.

En otras palabras el bloque hotfix es un área de almacenamiento temporal de sectores dañados.

15.7.3.5.1 - El “bit sucio”

Existen otros tipos de fallas que no pueden ser previstas por un sistema de archivos, algunas de ellas son:

- Fallas de energía
- Fallas causadas por Virus
- Reseteo accidental del equipo (no se actualiza la FAT del disco con respecto a la FAT que esta en memoria, en especial los datos que se encuentran en cache)

Por este motivo, en el SpareBloque se mantiene el DirtyFS (flag sucio).

Este bit es apagado cuando el sistema se cierra normalmente y esta consistente.

Cuando el sistema arranca, se verifica este flag y si esta prendido se ejecuta el CHKDSK, de manera de reconstruir las partes dañadas que sean necesarias.

Para facilitar el trabajo de reconstrucción y aumentar la robustez del sistema, HPFS mantiene información redundante de las partes criticas, por ejemplo utiliza listas doblemente encadenadas en lugar de listas regulares y también almacena en cada Fnode el principio del nombre del archivo o directorio que representa.

15.7.3.6. - **Atributos extendidos**

HPFS cuenta con un sistema de atributos dinámicos para cada archivo o directorio (EAs[5]).

En lugar de mantener una cantidad fija de bits estática asociada con cada archivo, se puede mantener un bloque dinámico de hasta 64k de información relacionada con los atributos del archivo.

Si se necesita mas que eso, se almacenaran los atributos en un árbol-b fuera del Fnode.

Los EAs mantienen los atributos convencionales de la FAT (Solo lectura, sistema, oculto, archivo) además de otros atributos respecto al tipo de archivo, que no mantiene la FAT. Por ejemplo el tipo de archivo y los permisos sobre el.

Esto posibilita además almacenar para cada archivo una lista de control de acceso (LCA) en caso de que el sistema operativo soporte esta funcionalidad.

15.7.3.7. - **Sistema de archivos instalable**

HPFS trae soporte para leer volúmenes de diferente estructura de otros sistemas operativos.

Para los volúmenes que no son HPFS se puede instalar en el sistema operativo el driver (controlador) que interpreta ese volumen de manera a las aplicaciones les resulte transparente acceder a volúmenes de diferente tipo.

[2] “rename” es una abreviatura del comando que se usa para modificar el nombre de un archivo o directorio.

[3] Spare en ingles significa repuesto.

[4] CHKDSK es la abreviatura de Check Disk, un utilitario de mantenimiento del disco. En Windows y Dos este utilitario lleva el mismo nombre.

[5] EAs es la abreviatura de Extended Atributes que en castellano significa atributos extendidos.

15.7.3.8. - **Comparación entre FAT y HPFS:**

A continuación se presentan las diferencias principales entre DOS y HPFS:

	FAT	HPFS
Longitud máxima de un nombre de archivo o directorio	11 en formato 8.3	254
Nro. de puntos (.) máxima permitida en un nombre	1	muchos
Atributos de un archivo	Bitmap estático	Bitmap estático mas 64k de datos dinámicos
Longitud máxima de una ruta	64	260
Mínimo espacio requerido para representar un archivo (sin contar los datos en si)	Una entrada de directorio (32 bits)	Una entrada (largo variable) + 512 bytes para el Fnode
Espacio promedio desperdiciado por archivo	½ cluster (típicamente 2048 bytes o mas)	½ sector (256 bytes)
Mínima unidad de almacenamiento	Cluster (típicamente 4096 bytes o mas)	Sector (512 bytes)
Espacio ocupado para la FAT (o mapa de archivos)	Centralizado en el track central del disco	Localizado cerca del Fnode de cada archivo
Información del espacio libre	Centralizado en el track central del disco	Localizado cerca de los bitmap's de espacio libre
Espacio libre representado por byte	2048 bytes (½ cluster con 8 sectores por cluster)	4096 bytes (8 sectores)
Estructura de directorio	Lista encadenada, búsqueda secuencial	Árbol-b, búsqueda binaria y agrupada.
Ubicación del directorio	Directorio raíz en el track central, el resto esparcido	Localizado cerca del volumen central
Estrategia de reemplazo de cache	LRU	LRU modificada, sensitiva al tipo de datos y datos históricos
Lectura adelantada	No hasta DOS4. El DOS4 y posteriores implementan una versión primitiva de esta técnica	Lectura adelantada sensitiva al tipo de datos y datos históricos
Escritura en segundo plano	No	Si

15.7.3.9. - **Resumen**

HPFS resuelve todos los históricos problemas del sistema FAT. Alcanza un excelente desempeño para el peor caso tanto para muchos archivos chicos como para algunos archivos grandes gracias a sus estructuras de datos avanzadas y a sus técnicas de lectura adelantada y de escritura en segundo plano.

El espacio en disco se utiliza en forma económica porque se puede "direccionar" a sector.

Las aplicaciones viejas que trabajan con nombre de archivos cortos son compatibles con HPFS y pueden modificarse fácilmente para aprovechar las ventajas de los "nombres largos".