

## CAPÍTULO II

# El sistema de archivos

### RESUMEN

Este capítulo se basa en el concepto abstracto de archivo para explicar la forma como se organiza la información sobre los soportes de almacenamiento externos. Se ofrece una visión general de los atributos que se asocian a los archivos, así como las operaciones que se pueden efectuar sobre ellos y la manera como se organizan en directorios jerárquicos. Se presenta la arquitectura general del sistema de archivo y se analiza el concepto abstracto de bloque como unidad mínima de almacenamiento y de manejo de los archivos. Se ofrecen detalles de los sistemas de archivos de colocación o asignación (contigua, enlazada e indexada), analizando las ventajas y desventajas de cada uno de ellos. También se discuten distintos métodos generales para controlar el espacio libre y se explican las particularidades de los sistemas de archivos que están asociados a los sistemas operativos Windows (FAT y NTFS) y a los sistemas operativos tipo UNIX (UFS, ext, ext2, ext3 y ext4). El capítulo finaliza con un resumen y una sección de ejercicios propuestos.

**Palabras clave:** sistema de archivos, archivos, directorios, bloques, métodos de asignación de espacios, control de espacio libre.

---

*¿Cómo citar este capítulo? / How to cite this chapter?*

Lezcano-Brito, M. G. (2017). El sistema de archivos. En *Fundamentos de sistemas operativos. Entornos de trabajo* (pp. 77-104). Bogotá: Ediciones Universidad Cooperativa de Colombia.



## CHAPTER II

# File system

### ABSTRACT

This chapter is based on the abstract concept of file to explain how information is organized on external storage media. It provides an overview of the attributes associated with files, as well as the operations that can be performed on them and how they are organized into hierarchical directories. The general architecture of file systems is presented and the abstract concept of block as the minimum file storage and management unit is analyzed. Details of placement and allocation (contiguous, linked and indexed) file systems are given, evaluating the advantages and disadvantages of each of them. Different general methods to control free space are also discussed and the peculiarities of file systems that are associated with Windows operating systems (FAT and NTFS) and UNIX operating systems (UFS, ext, ext2, ext3 and ext4) are explained. The chapter ends with a summary and a section of proposed exercises.

**Keywords:** File system, files, directories, blocks, space allocation methods, free space control.

Desde el punto de vista de los usuarios finales, la parte más “visible” de cualquier SO es su **sistema de archivo**. Este fenómeno ocurre debido a que la mayoría de los usuarios hace trabajos que tienen efectos directos en este subsistema y los realizan, además, de forma interactiva.

Se puede añadir que la mayoría de las aplicaciones también actúa con el sistema de archivo si se toma en cuenta que su propósito principal es procesar datos, los cuales muchas veces (en realidad, casi siempre) residen en archivos y el resultado de procesar esos datos se almacena en archivos. El fenómeno es tan común que algunas personas tienen una visión simplificada del SO al reducirlo al sistema de archivo.

El sistema de archivo está formado por dos partes perfectamente distinguibles (al menos en funciones): un **conjunto de archivos** que almacenan los datos en sí y una estructura de datos, conocida como **directorio**, que permite localizar la información contenida en los archivos. Ambos aspectos, entre otros, se tratan en este capítulo.

## II.1 EL CONCEPTO DE ARCHIVO

Las computadoras pueden almacenar información permanente en diferentes soportes, tales como discos (de varios tipos), cintas magnéticas, memorias *flash*, etc. La naturaleza de cada uno de estos elementos de almacenamiento es disímil, y para poder trabajar con eficiencia, debe existir una forma homogénea de tratar sus contenidos. El SO hace una abstracción de los elementos físicos que se relacionan con cada uno de los equipos de almacenamiento y trata los datos como una **unidad lógica** que recibe el nombre de **archivo**.

Un archivo es un conjunto de datos que tienen un formato común y se agrupan bajo un nombre. Ese nombre se usa para hacer operaciones sobre el archivo, las cuales pueden ser: borrarlo, copiarlo, moverlo, renombrarlo, leerlo, etc.

Cuando se dice que el conjunto de datos tiene un formato común, se está estableciendo una forma para organizar el archivo, que puede ser una simple cadena de bytes (como lo hace Unix) o un conjunto de registros.

## II.2 LA ARQUITECTURA DEL SISTEMA DE ARCHIVO

Una forma típica de organizar el sistema de archivo se presenta en la figura II.1 (puede diferir de acuerdo con las particularidades del SO). En ella, el sistema se presenta como un conjunto de capas: las capas inferiores están más cercanas al *hardware* y las

superiores están más cercanas a los usuarios. Esta organización hace que se alcancen mayores niveles de abstracción en las capas superiores, lo que permite que los usuarios se aíslen de las especificidades y complejidades asociadas a los equipos periféricos.

La capa de los **manipuladores de equipos** (*device driver*) es la que actúa directamente con el *hardware*, es decir, los periféricos de entrada/salida, como los canales o los controladores. La capa es responsable de iniciar la operación de entrada/salida (E/S) sobre el equipo y completarla.

Entrada/salida lógica
Supervisor de E/S básico
Sistema de archivo básico (E/S física)
Manipuladores

Figura II.1. Organización típica de un sistema de archivos. Fuente elaboración propia.

El **sistema de archivo básico** es el primer enlace entre el sistema de cómputo y el mundo externo. A este nivel, se trabaja con bloques de datos (unidades de información de un cierto tamaño) que se intercambian con el equipo y se pasan desde o hacia la memoria. En esta capa, no se conoce nada acerca de la estructura que tienen los datos que se manipulan.

El **supervisor de E/S básico** es el responsable de comenzar y culminar las entradas y salidas sobre los archivos. Ya en este nivel se usa una estructura de control que permite conocer la situación de los equipos y el estado de los archivos que se almacenan en ellos, entre otros pormenores.

La capa de **entrada/salida lógica** es la responsable de presentarles a los usuarios y a las aplicaciones una interfaz para acceder a la información que se almacena en los equipos. En esta capa, se trabaja con archivos y no con bloques, como lo hace la capa de entrada/salida física; este es el nivel que “ven” los usuarios finales y según su punto de vista este es el único nivel que existe, ya que ellos realizan sus operaciones sobre el concepto abstracto de archivo (copian o mueven archivos, ordenan su ejecución, etc.).

### II.3 ATRIBUTOS DE LOS ARCHIVOS. OPERACIONES SOBRE ELLOS

Asociado a todo archivo existe un conjunto de atributos que puede variar de un SO a otro (aunque algunos son invariantes). Entre esos atributos cabe mencionar los siguientes:

- \* El nombre del archivo. Identifica unívocamente al archivo dentro de la estructura de directorio.
- \* El tipo de archivo. Se refiere al formato de los archivos, que puede ser de texto, binario, de enlace simbólico, etc.
- \* Datos acerca de la creación o modificación del archivo, tales como fecha, hora, etc.
- \* El tamaño del archivo, puede estar especificado en bytes, bloques, etc.
- \* Atributos de protección. Especifican qué se puede hacer con el archivo y quién puede hacerlo, por ejemplo: leerlo solamente, de lectura/escritura, quién es el dueño, etc.

Como ya se ha apuntado, sobre los archivos se pueden realizar diversas operaciones; la mayoría de esas operaciones necesita buscar información en el **directorio**.

Antes de acceder a la información contenida en un archivo es necesario abrirlo, para lo cual se busca su nombre en los directorios. Una vez localizado el archivo, se pone una referencia a él dentro de una estructura de datos que se denomina **tabla de archivos abiertos**, la cual permite conocer cuál es la próxima entrada del archivo que se leerá en acceso secuencial, quién o quiénes pueden hacer esa operación, etc. La tabla de archivos abiertos permanece siempre en memoria y la referencia citada estará en ella hasta que se cierre el archivo.

Dentro de la tabla de archivos abiertos, los archivos se referencian por números, conocidos como manipuladores (*handles*), o sea, en esa tabla los archivos “pierden” su nombre temporalmente.

```

$ ls -l
total 20
- rwx---- madiedo Master 1024 jul 8 12:49 cpu.c
d rwxr-xr-x lezcano root 4096 jun 1 20:44 trabajo
l rwx---- madiedo Master 512 sep 2 12:56 archivos
- rwx---- lezcano root 1024 sep 1 23:36 file1
- rwx---- lezcano root 4096 sep 214:53 clase1

```

Tipo	protección	propietario	grupo	tamaño	fecha y hora	nombre
-	rwx----	madiedo	Master	1024	jul 8 12:49	cpu.c
d	rwxr-xr-x	lezcano	root	4096	jun 1 20:44	trabajo
l	rwx----	madiedo	Master	512	sep 2 12:56	archivos
-	rwx----	lezcano	root	1024	sep 1 23:36	file1
-	rwx----	lezcano	root	4096	sep 214:53	clase1

Figura II.2. Algunos de los atributos de los archivos en un SO Unix. Fuente elaboración propia.

La figura II.2 muestra el resultado de ejecutar el comando `ls -l`<sup>28</sup> en un SO de la familia Unix. Pueden observarse algunos de los atributos de los archivos, tales como: su tamaño, la fecha de modificación, el propietario, etc. Otros atributos, como la referencia al **nodo-i** (se explica más adelante), permanecen ocultos al usuario. En este caso, la primera columna del listado especifica el tipo de archivo de acuerdo con el siguiente convenio:

- \* -, representa un archivo regular.
- \* d, especifica que el archivo es un directorio.
- \* l, especifica un enlace (*link*) simbólico, etc.

La segunda columna es una tira de nueve bits que establece los permisos sobre el archivo. La tira se divide en tres ternas de bits: la primera terna especifica los permisos para el dueño del archivo (*owner*), la segunda establece los permisos para los miembros del grupo (*group*) y la tercera fija los permisos para los otros usuarios (*others*). Si todos los bits de una terna tienen el valor 1, externamente se ven en la forma `rwX`, donde: `r` especifica permiso de lectura, `w` especifica permiso de escritura y `X` especifica permiso de ejecución; el símbolo - en la posición de un elemento significa que el bit tiene el valor cero y que no se tiene el permiso que esa posición especifica.

<sup>28</sup> `ls -l`. Comando de Unix para listar (*list*) en formato largo (*long*).

## II.4 ESTRUCTURA DE DIRECTORIO

Todo sistema de archivo posee, como parte de su organización, una estructura de datos denominada **tabla de directorio** (puede que tenga otro nombre pero con la misma funcionalidad), que sirve para localizar los archivos. Todos los medios de almacenamiento poseen esa estructura de datos, que se construye cuando se le da formato al soporte.

La forma que tenga el directorio depende del SO; la figura II.3 presenta una tabla de directorio típica de un SO hipotético. Como se puede apreciar, en esta estructura de datos están presentes diversos campos para especificar datos acerca de los archivos (el encabezado de la primera fila de la tabla solo tiene un fin didáctico, pero no forma parte de ella).

NOMBRE	TIPO	DIRECCIÓN DE INICIO	FECHA	HORA	TAMAÑO	PROPIETARIO

Figura II.3. Tabla de directorio de un SO hipotético. Fuente elaboración propia.

Algunos sistemas de archivos no poseen mucha información en el directorio. Se pueden tener dos extremos: en uno el directorio solo contiene un nombre y un puntero a otra estructura que complementa la información (por ejemplo, Unix); en el otro extremo, el directorio contiene toda la información relativa al archivo, incluso su localización completa (el ya obsoleto CP/M).

La estructura del directorio puede ser de un solo nivel (casi no se usa hoy en día) o de varios niveles o jerárquica. La figura II.4 muestra la estructura de directorio típica de un SO de la familia Unix.

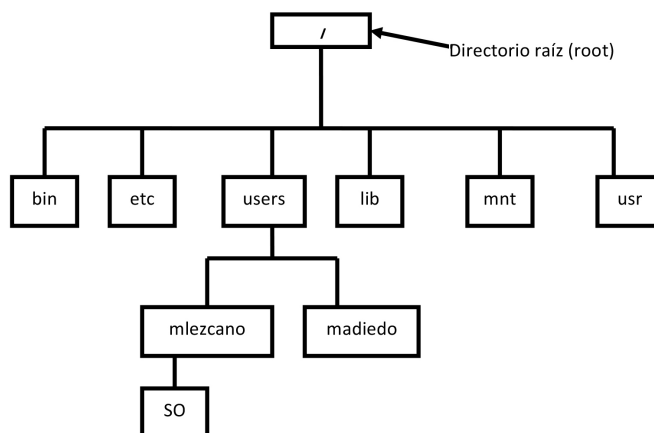


Figura II.4. Estructura de directorio típica de Unix. Fuente elaboración propia.

Desde el punto de vista de los usuarios y las aplicaciones, existen dos formas de acceder a la información contenida en una estructura como la mostrada en la figura II.4:

\* Camino o ruta absoluta

Especifica el camino o la ruta que conduce hasta el directorio o archivo que se desea localizar, tomando como punto de partida el directorio raíz. Por ejemplo, los caminos: /users/mlezcano y /users/madiedo trazan la ruta desde el directorio raíz (se simboliza con el signo / al inicio de la cadena) hasta los directorios mlezcano y madiedo, respectivamente.

\* Camino o ruta relativa

Especifica el camino que conduce hasta el directorio o archivo que se desea localizar a partir de algún directorio diferente al directorio raíz. Por ejemplo, el camino mlezcano/SO que comienza en el directorio mlezcano.

## II.5 EL CONCEPTO DE BLOQUE

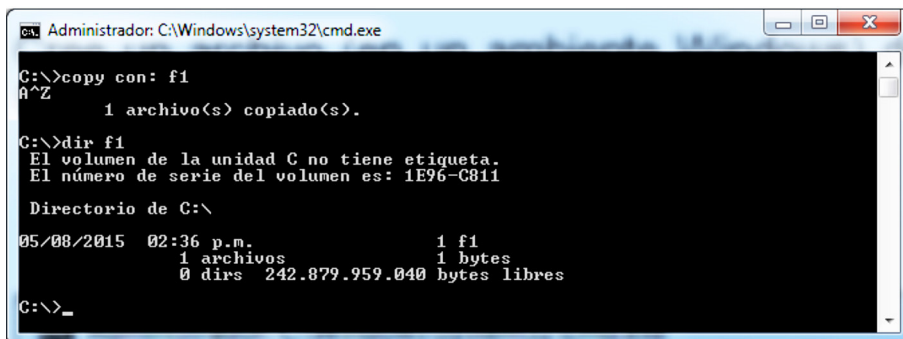
Un **bloque es la unidad mínima de asignación de espacio** en un soporte específico. Esto quiere decir que, siempre que se pida un cierto espacio en un soporte de almacenamiento, el sistema de archivo proporciona una cantidad que, por lo regular, es mayor que la que se pide.

El problema de asignar el espacio por bloques viene dado por el hecho de que sería muy poco práctico que cada byte del sistema de archivo fuera accedido de forma individual, dado que se gastaría más espacio para controlarlo que para guardar información útil.

Esta forma de asignar espacio hace que se pierdan algunos bytes, a veces todos menos uno, en el último bloque del archivo. Este fenómeno se conoce como **fragmentación interna** y no tiene solución.

El problema de la fragmentación interna en los archivos es fácil de apreciar. Con tal propósito, realice el siguiente ejercicio:

Cree un archivo (en un ambiente Windows) como se muestra en la figura II.5. El archivo se crea con un solo carácter (la letra A), pues ni siquiera se ha añadido cambio de línea después de ese carácter (^z es decir, control Z, es el fin de archivo en los SO de la familia Windows). Después, se usa el comando dir para que muestre el tamaño del archivo que, como es lógico, tiene un solo byte.



```

C:\>copy con: f1
A^Z
        1 archivo(s) copiado(s).

C:\>dir f1
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 1E96-C811

Directorio de C:\

05/08/2015  02:36 p.m.                1 f1
               1 archivos              1 bytes
               0 dirs  242.879.959.040 bytes libres

C:\>_
  
```

Figura II.5. Creación de archivo desde el intérprete de comandos de Windows. Fuente: Captado desde el SO Windows.

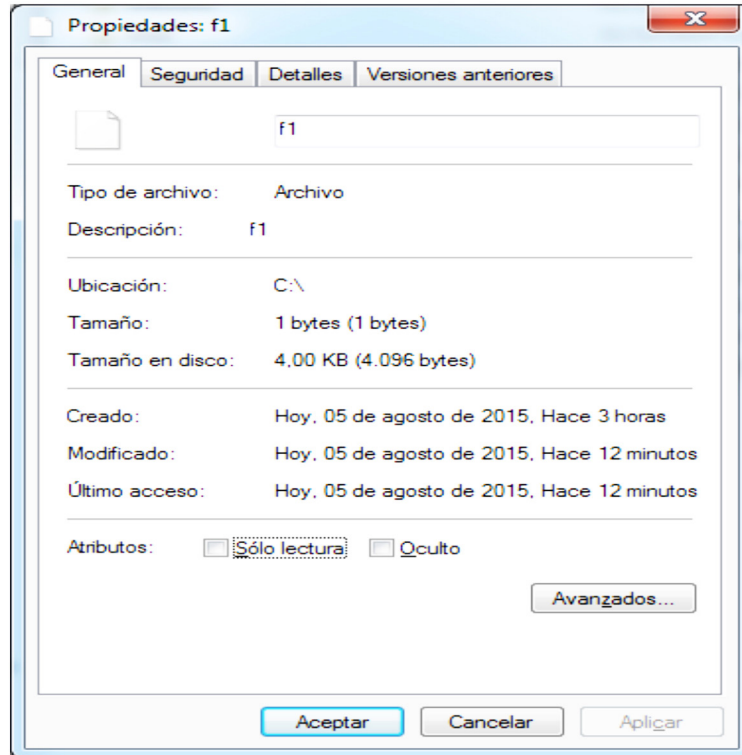


Figura II.6. Muestra de fragmentación interna. Fuente: Captado desde el SO Windows.

Ahora vaya al ambiente gráfico y oprima el botón derecho del ratón encima del nombre del archivo f1. Observe la figura II.6, que debe ser similar a la que usted obtenga; puede notarse que el tamaño del archivo es un byte pero en el disco ocupa 4096 bytes, lo cual viene dado porque el tamaño de los bloques en ese disco es de 4096 bytes y el SO no puede asignar espacio menor que un bloque; por tanto, de los 4096 bytes asignados al archivo f1, solo se utiliza un byte y el resto, aunque pertenece a él, no lo necesita (al menos por el momento) provocando una fragmentación interna de 4095 bytes.

En otra unidad, la fragmentación podría ser mayor o menor porque cada una tendrá un tamaño de bloque que normalmente es mayor mientras mayor sea su capacidad, aunque ese tamaño se puede cambiar cuando se formatea la unidad. El comando `chkdsk`, que se usa para hacer verificaciones acerca de la integridad de la unidad, también reporta el tamaño de los bloques.

La información acerca del tamaño de los bloques, entre otras, está contenida en una parte del soporte de información que el SO reserva para su trabajo.

## II.6 ASIGNACIÓN DE ESPACIO

Existen diferentes métodos y formas para asignar el espacio en un soporte de almacenamiento externo. Para considerar estas formas se deben tomar en cuenta los siguientes criterios:



1. Cuando se cree un nuevo archivo, ¿se le asignará todo el espacio que necesita?
2. ¿Qué tamaño debe tener un bloque?

Si a la primera pregunta se responde que sí, habrá que asignarle todo el espacio que necesitará el archivo cuando se cree, pero es difícil (e incluso imposible) conocer esa información en ese momento, lo cual puede derivar en la sobrestimación de necesidades (malgastando espacio). Por ese motivo, es mejor realizar asignaciones dinámicas, lo cual consiste en hacer crecer el archivo pidiendo bloques nuevos cada vez que se necesite más espacio.

Para la segunda pregunta, debe analizarse que:

- \* Un tamaño de bloque muy pequeño exige disponer de una tabla enorme para controlar los bloques, es decir, para llevar el control de cuáles están ocupados y cuáles están libres.
- \* Un tamaño muy grande del bloque gasta mucho espacio, dado que el último bloque de cada archivo puede quedar prácticamente vacío.

A partir del análisis anterior, se infiere que hay que tomar una posición intermedia, en la cual los bloques sean lo suficientemente pequeños para no provocar tanta fragmentación interna y lo suficientemente grandes para no tener que usar demasiados recursos para controlarlos. Queda claro que no es tan fácil definir el tamaño “ideal” de un bloque.

#### II.6.1 Tipos de asignación

Existen tres formas básicas para asignar espacio en equipos de almacenamiento externo. En este apartado, se analiza cada una de ellas destacando sus ventajas y desventajas.

##### Asignación contigua

Este tipo de estrategia consiste en asignar un conjunto de bloques contiguos en el momento de la creación del archivo, como se aprecia en la figura II.7.

Obsérvese que: el archivo file1 comienza en el bloque 1, mientras que el archivo file2 comienza en el bloque 4, ambos números actúan como una especie de apuntador al primer bloque del archivo; la figura II.7 usa flechas discontinuas para resaltar esa idea.

En los sistemas de asignación contigua, solo se necesita conocer el **bloque de inicio** y la **cantidad de bloques** que ocupa cada archivo, debido a que los bloques son consecutivos y están en orden a partir del primero que se especifica en la tabla de directorio. En el ejemplo, el archivo file1 está formado por tres bloques que son: 1, 2 y 3; mientras que los dos bloques del archivo file2 son el 4 y el 5.

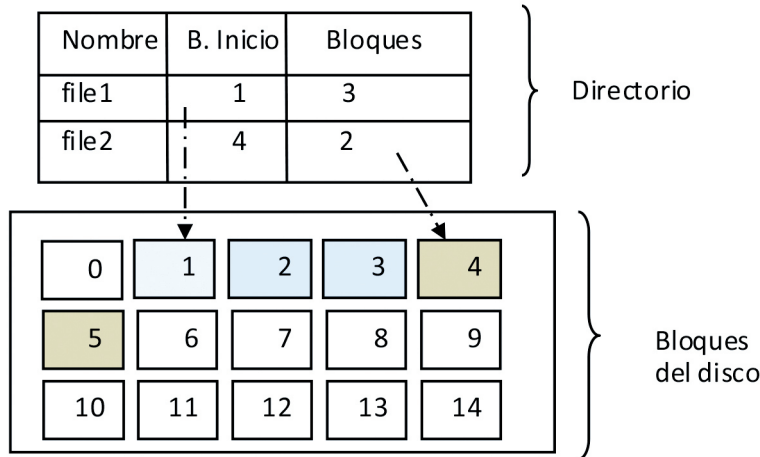


Figura II.7. Asignación contigua. Fuente elaboración propia.

#### Ventajas:

1. Permite el **acceso directo** a los datos, para lo cual solo se necesita la dirección de inicio del archivo (está contenida en la tabla de directorio) y el desplazamiento a partir de esa dirección.
2. En los discos, el acceso es más rápido debido a que se efectúan menos movimientos mecánicos del cabezal de lectura/escritura para acceder a los datos (están unos a continuación de otros).
3. La recuperación de datos perdidos en algún borrado es más fácil y hay mayores garantías de que sea exitosa.

#### Desventajas:

1. Un archivo solo puede crecer hasta el inicio de su vecino, ya que cualquier otro "hueco" que esté libre no estará contiguo a él. Este problema hace que los usuarios tiendan a sobrestimar la longitud de los archivos y reserven espacios que en realidad son mayores que los que necesitan, lo que implica un malgasto de espacio. En el ejemplo de la figura II.7, el archivo file1 no puede crecer debido a que está acotado por el archivo file2.
2. Provoca **fragmentación externa**, debido a que puede que no sea posible satisfacer una solicitud de espacio a pesar de existir el espacio en forma de "huecos", o espacios no contiguos, y que sumados satisfacen la petición. La mayoría de los SO que usan este tipo de asignación proveen algún mecanismo de desfragmentación que tiene como fin reunir todos los huecos en uno solo en forma contigua, pero este mecanismo es costoso ya que implica la detención de todos los trabajos que se realizan sobre el periférico.

A partir de este análisis, se concluye que las desventajas de la asignación contigua tienen un peso mayor que sus ventajas, de ahí que no sea muy utilizada actualmente, a no ser en soportes cuya naturaleza permita almacenar grandes archivos por tiempos prolongados con el propósito de resguardarlos.

### **Estrategias para la asignación de espacio en asignación contigua**

Para satisfacer una petición de tamaño  $m$ , el sistema de archivo tiene que encontrar un espacio de tamaño  $n$  que cumpla que  $m \leq n$ . Existen tres estrategias o políticas para tomar la decisión respecto a cómo escoger ese espacio:

1. El primer acceso. Significa tomar el primer espacio que satisfaga la demanda, es decir que cumpla que  $m \leq n$ .
2. El mejor acceso. Significa tomar un espacio que satisfaga la demanda y que sea el menor de todos los posibles.

Es decir,  $m \leq n$  y  $n = \text{menor } h_i \text{ de } H = \{h_1, h_2, \dots\}$ , donde  $H$  representa el conjunto de todos los espacios libres.

3. El peor acceso. Significa tomar un espacio que satisfaga la demanda y que sea el mayor de todos los posibles.

O sea,  $m \leq n$  y  $n = \text{mayor } h_i \text{ de } H = \{h_1, h_2, \dots\}$ .

Debe observarse que la estrategia del mejor acceso tiende a dejar pequeños espacios dispersos (se dice que el medio está fragmentado). Esos pequeños espacios muchas veces no son útiles a ninguna demanda, de forma que el “mejor acceso” puede resultar siendo la peor solución.

De otra parte, la estrategia del peor acceso deja espacios mayores que, en general, tendrán mayor probabilidad de ser útiles.

En relación con el primer acceso, es totalmente impredecible el resultado ya que dependerá del orden en que estén los espacios libres.

### **Asignación enlazada**

La asignación enlazada trata de resolver los problemas del método anterior. En este esquema, cada bloque contiene un puntero al próximo bloque y los bloques pueden estar en cualquier parte del equipo de almacenamiento.

Observe la idea esquematizada en la figura II.8. Cada bloque tiene dos partes: la primera está destinada a datos y la segunda contiene un puntero. En el caso que se analiza en la figura II.8, el directorio especifica que el primer bloque del archivo file1 es el 2 (los números de los bloques comienzan en cero); el primer campo del bloque 2 contiene el dato  $d_1$  y su puntero apunta al bloque 5 (las líneas discontinuas son para reafirmar la idea); el primer campo del bloque 5 contiene el dato  $d_2$  y su puntero apunta al bloque 7, el cual contiene el dato  $d_3$  y su puntero es nulo, lo que significa que ese es el último bloque del archivo file1 que está compuesto por los bloques 2, 5 y 7.

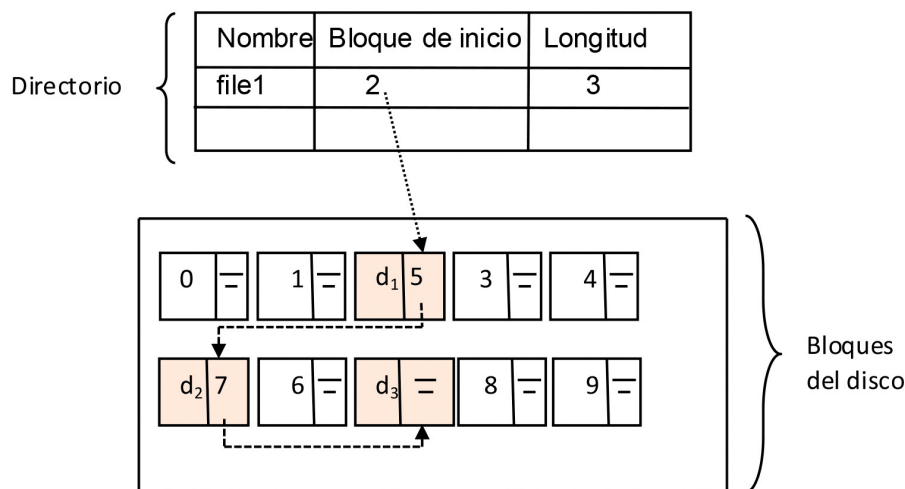


Figura II.8. Asignación enlazada. Fuente elaboración propia.

Ventajas:

1. Como los bloques de un archivo pueden estar dispersos por todo el equipo de almacenamiento, los archivos podrán crecer mientras existan bloques libres.
2. Elimina el problema de la fragmentación externa.

Desventajas:

1. No permite el acceso directo; para acceder al bloque  $n$  hay que recorrer los  $n-1$  bloques que le preceden.
2. La dispersión de los bloques hace que el acceso sea más lento en general.
3. Gasta espacio adicional debido a que cada bloque contiene un campo dedicado al puntero, donde no se guardan datos útiles desde el punto de vista del usuario.

Este tipo de asignación, al tratar de resolver los problemas de la anterior, empeora la situación dado que un sistema de archivo que solo permita acceso secuencial a sus datos se hace extremadamente lento e ineficiente.

### Asignación indexada

La idea básica se aprecia en la figura II.9. Obsérvese que cada archivo usa uno de los bloques del volumen (el 4 en el ejemplo) para almacenar las localizaciones de los bloques que pertenecen al archivo. Ese bloque, que contiene las direcciones de los demás bloques del archivo, es el **bloque de índices** del archivo y desde la tabla de directorio se apunta a él. En este caso (al igual que en la asignación enlazada), no es necesario conocer la longitud del archivo para establecer dónde termina (aunque sí es bueno que esa información esté disponible para otros fines).

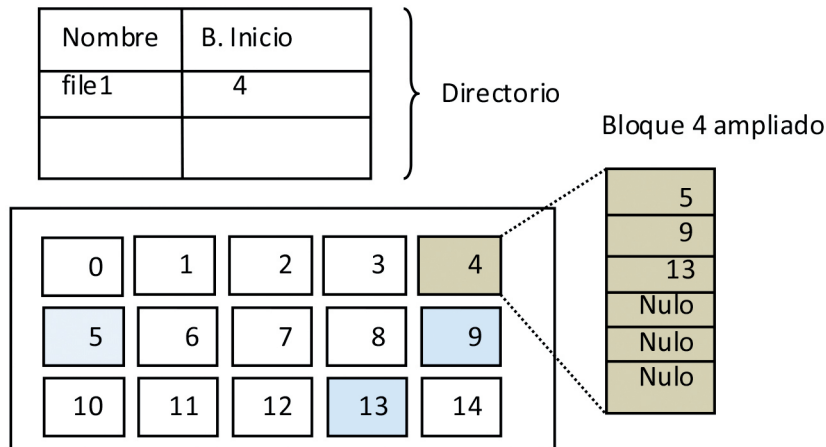


Figura II.9. Asignación indexada. Fuente elaboración propia.

En el caso de la figura II.9, el bloque 4, que es el bloque de índices, no contiene datos sino punteros a otros bloques. Por ese motivo, se conoce que el archivo file1 está compuesto por los bloques 5, 9 y 13. Debe observarse que el bloque 4 lo usa el SO para localizar los bloques del archivo y, por tanto, no contiene ningún dato del archivo en sí.

La solución que ofrece el SO Unix es prácticamente así, mientras que la que ofrece la familia Windows, con su sistema de archivo FAT<sup>29</sup>, difiere un tanto dado que existe una gran tabla de índices para todos los archivos del sistema de archivo de un volumen dado. De hecho, dentro de la FAT existe una lista enlazada que es necesario recorrer para localizar los bloques de datos, de ahí que algunos autores la clasifiquen como asignación enlazada cuando en realidad no lo es.

La asignación indexada es la más utilizada actualmente, debido a que resuelve los problemas críticos. Entre sus ventajas se pueden enumerar las siguientes:

1. No provoca fragmentación externa.
2. Permite el acceso aleatorio.
3. Los archivos pueden crecer mientras haya espacio y forma de hacer referencia a sus bloques.

Las malas noticias son que el contenido de un archivo, por lo general, está disperso, lo cual hace que los accesos sean más lentos que en la asignación contigua, pero las desventajas de esta última son demasiado serias como para no tomarlas en cuenta.

## II.7 CONTROL DEL ESPACIO LIBRE

Para controlar el espacio libre, se pueden usar varios tipos de estructuras de datos. El sistema de archivo FAT usa la misma estructura para referirse a los espacios libres y

<sup>29</sup> Tabla de asignación de archivos (File Allocation Table).

a los bloques de los archivos, pero en realidad, y pese a la popularidad de los SO de la familia Windows, esa forma de llevar el control de espacios libres no es la más eficiente.

### Control de espacio libre a través de una tabla

El espacio libre se puede controlar con una **tabla de espacios libres** que tenga dos campos: el primero contiene la dirección del primer espacio libre y el segundo contiene su tamaño. Observe la figura II.10.

DIRECCIÓN DE INICIO	TAMAÑO
10	23
80	12

Figura II.10. Tabla de espacios libres. Fuente elaboración propia.

Cuando se necesita un espacio, el sistema de archivo consulta esa tabla y toma el que se necesita; después, debe actualizar la tabla eliminando la entrada que tomó (si es que se ocupó totalmente) o restableciendo el inicio y el tamaño si solo se tomó parte de la entrada. Este esquema se usa en algunos sistemas de asignación (colocación) contigua y tiene alguna sobrecarga, dado que cada vez que se libere un espacio no basta con ponerlo en la tabla de espacios libres, sino que habrá que verificar si existía algún espacio antes y después de él, los cuales deben eliminarse de la tabla para “unirlos” en un solo hueco.

### Control del espacio libre usando un mapa de bits

Una mejor solución es tener un **mapa de bits** (figura II.11), que tendrá una longitud igual a la cantidad de bloques que tenga el volumen, de modo que a cada bit del mapa corresponde a uno y solo un bloque del volumen. Se establece un convenio, la mayoría de las veces, un 0 en la posición  $n$  del mapa de bits significa que el bloque  $n$  está libre y un 1 significa que está ocupado.

Para satisfacer una solicitud de  $n$  bloques, habrá que buscar  $n$  bits del mapa de bit con valor 0.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0

Figura II.11. Mapa de bits para control de espacios libres. Fuente elaboración propia.

En la figura II.11, los bloques 3, 5, 6, 7, 10, 11, 12, 13, 14 y 19 están libres y los restantes están ocupados. En caso de que se fuera a satisfacer una petición que necesitara 10 bloques libres, ocurriría lo siguiente:

- \* En un sistema de archivo de colocación enlazada o indexada, se podría satisfacer debido a que existe esa cantidad de bloques libres (no importa si están o no contiguos).

- \* En un sistema de archivo de colocación contigua, esa petición no podría satisfacerse debido a que la máxima cantidad de bloques libres en forma contigua que existe es cinco. Aunque existen cuatro “huecos” libres que sumados dan 10: el primero, que tiene un solo bloque (el 3); el segundo, que tiene tres bloques (5, 6 y 7); el tercero, que tiene cinco bloques (10, 11, 12, 13 y 14); y el último, que tiene un bloque (el 19). En este caso, se aprecia el fenómeno de la fragmentación externa.

### **Desfragmentar**

En los sistemas de archivos contiguos, es necesario desfragmentar cuando el espacio libre queda constituido por “huecos” de diferentes tamaños que no satisfacen las solicitudes. En ese caso, la desfragmentación (para muchos autores es una compactación) se refiere al hecho de agrupar todo el espacio libre en un solo hueco.

En los sistemas enlazados e indexados, no es necesario desfragmentar, debido a que los espacios libres se ven como bloques individuales que pueden usarse de cualquier forma, dado que los archivos pueden estar en bloques dispersos. Cuando los bloques están dispersos, existe la necesidad de realizar múltiples movimientos del cabezal de lectura/escritura (cuando es un disco), lo que resulta en accesos más lentos. Para ayudar a resolver ese problema, la mayoría de los SO que tienen sistema de archivos indexados ofrece programas especiales para desfragmentar los soportes de información. En este caso, la desfragmentación es la acción de hacer que todos los archivos (si es que es posible) estén en forma contigua.

## **II.8 EJEMPLOS DE SISTEMAS DE ARCHIVOS**

En esta sección, se describen varios sistemas de archivos de diferentes SO. El objetivo es analizar cómo se instrumentan las ideas generales que se han discutido hasta el momento.

### **II.8.1 Sistemas de la compañía Microsoft**

Los SO de la compañía Microsoft disponen de dos sistemas de archivos: el FAT y el NTFS.

#### **El sistema de archivo File Allocation Table (FAT)**

El sistema de archivo FAT fue el primero usado por la compañía Microsoft y lo instrumentó sobre el SO MS-DOS (Microsoft Disk Operating System). El sistema aún se usa, sobre todo en soportes como las memorias *flash*. Aunque se puede usar en discos duros, debe tomarse en cuenta que no es un sistema seguro.

La figura II.12 muestra las dos estructuras de datos que soportan el sistema de archivo FAT. En la parte superior, se observa la tabla de directorio, mientras que la parte inferior muestra la estructura de datos que da nombre al sistema y que se conoce como FAT.

NOMBRE	EXTENSIÓN	ATRIBUTOS	RESERVADO	HORA	FECHA	CLÚSTER INICIAL	LONGITUD
SO. doc						12	
Prueba						23	

	0	1	2	3	4	5	6	7	8	9
0	0	0	eof	8	0	0	0	0	2	0
1	0	0	3	0	0	0	0	0	0	0
2	0	0	0	30	0	0	0	0	0	0
3	31	38	0	0	0	0	0	0	50	0
4	0	0	0	0	0	0	0	0	0	0
5	eof	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

Figura II.12. Sistema de archivo FAT original. Fuente elaboración propia.

La tabla de directorio de este sistema de archivos brinda los siguientes datos acerca de los archivos: su **nombre**, la **extensión** (forma parte de la identificación del archivo y se asocia a su contenido), los **atributos** (oculto, de lectura solamente, sistema y archivo), la **hora y fecha** de actualización, su **longitud** y un campo muy especial para especificar cuál es el **bloque<sup>30</sup> inicial** del archivo, es decir, el primer bloque del archivo.

La tabla FAT se localiza muy cerca del inicio del volumen y en un lugar fijo (son dos copias por si se daña una). El directorio raíz también se localiza en un lugar fijo, todo lo cual se establece cuando se formatea el volumen.

Obsérvese que la tabla de directorio que usó inicialmente este sistema de archivo también dejó un campo reservado para futuras ampliaciones. Cuando el sistema FAT quiso extender los nombres de los archivos más allá de la convención 8+3 (ocho caracteres para el nombre y tres para la extensión), la práctica demostró que el espacio reservado era muy pequeño.

La estructura de datos FAT tiene un doble propósito: el primero es conocer la localización de los bloques de datos de cada archivo y el segundo es controlar el espacio libre.

En el ejemplo presentado, la tabla de directorio indica que el bloque inicial del archivo SO.doc es el 12, esa especie de puntero le sirve al SO para buscar la posición 12 en la FAT. La posición 12 de la FAT contiene un número que indica el próximo bloque de ese

<sup>30</sup> Clúster según la terminología Windows.



archivo (es el 3); y a su vez, la entrada 3 de la FAT sirve como nuevo puntero y señala al clúster 8 como el próximo, el cual señala al 2 y en la posición 2 de la FAT se ha situado un número especial que marca el fin del archivo (representado simbólicamente por *eof-end of file*). Es decir, los bloques sobre los que reside el archivo SO.doc son: 12, 3, 8 y 2 (en ese orden). De igual forma, el archivo prueba comienza en el bloque 23 y termina en el 50.

La FAT original dejó los dos primeros clústeres del disco para uso del SO (observe de nuevo la figura II.12)

Ejercicio:

- \* Encuentre la secuencia de bloques que forman parte del archivo prueba.

Las posiciones de la tabla FAT que tienen un cero especifican que los bloques referidos están libres. Debe quedar claro que el sistema FAT no es más que una tabla de apuntadores a bloques y el SO tendrá que hacer los cálculos para saber en qué pista y sector del disco (por ejemplo) está físicamente el bloque *n*; para ello, usa información que está al inicio del disco (sector del boot) y que especifica, entre otras cosas, el tamaño de los bloques.

### Sistema de archivo New Technology File System (NTFS)

El sistema NTFS también organiza los volúmenes en archivos y directorios, pero en esta forma organizativa no existe ningún objeto especial en el disco, ni ninguna localización específica (como el FAT). Además, no existe una dependencia en relación con el *hardware* como es la longitud de 512 bytes de los sectores.

Debe observarse que la existencia de algún objeto especial (como la FAT) hace que el daño en dicho objeto sea catastrófico para el sistema de archivo.

Los diseñadores del sistema de archivo NTFS se trazaron la meta de hacer un sistema confiable que soportara los requisitos del estándar POSIX y que dejara atrás las limitaciones del sistema FAT.

NTFS es un sistema de archivos recuperable debido a que mantiene un registro de las transacciones hechas en el sistema de archivos que se conoce como registro por (o de) diario (*journaling*).

El registro de diario es una bitácora que permite almacenar información para restablecer los datos afectados por las transacciones. Durante el tiempo que demora una transacción, se hace lo siguiente:

1. Se bloquean las estructuras de datos afectadas por la transacción. De esta forma, ningún proceso distinto al que hace la transacción puede modificar las estructuras de datos involucradas en la operación.
2. Se reserva un recurso para almacenar el diario (*journal*), por ejemplo, algunos bloques de disco. En esos bloques, se guarda el estado del volumen antes de la operación, de modo que si ocurre un fallo (de energía, del SO, etc.), se pueda regresar al pasado.
3. Se efectúan las modificaciones en las estructuras de datos (una a una) y para cada modificación:

- a. Se apunta en el diario cómo deshacerla, después de lo cual se realiza la modificación.
  - b. Si se cancela la transacción, se deshacen los cambios, uno a uno, y se borra la traza guardada en el diario.
4. Si se logra realizar la transacción, se borra el diario y se desbloquean las estructuras de datos.

La longitud de los archivos y volúmenes NTFS puede ser de hasta  $2^{64}$  bytes (16 exabytes)<sup>31</sup>. No es recomendable usar NTFS en volúmenes menores que 400 MB, debido a que la sobrecarga que agrega el sistema no se justifica para un volumen de esa dimensión.

En los volúmenes formateados con el sistema de archivo NTFS, se crean varios **archivos de sistema** que pueden alojarse en cualquier parte del volumen con el objetivo de que no suceda como en el sistema FAT, donde un daño en el área que ocupa la FAT hace que el sistema de archivo sea inaccesible.

Los archivos de sistema o *metafiles* son (entre otros):

- \* \$attrdef. Tabla de definición de atributos (*attributes definition*). Contiene todos los atributos del volumen (pueden ser del sistema o definidos por usuarios).
- \* \$badclus. Clúster dañados (*bad clusters*). Contiene la relación de los clúster dañados del volumen, los cuales no pueden usarse.
- \* \$bitmap. Mapa de bits de clúster asignados (*cluster allocation bitmap*). Contiene un mapa de bits del volumen, que especifica los clúster asignados y libres. En el sentido de mantener la relación de clúster asignados y libres, es el equivalente del FAT, pero no se usa para localizar los archivos.
- \* \$boot. Archivo boot (*boot file*). Si el volumen es cargable (“bootable”), contiene el programa de arranque (*bootstrap*).
- \* \$logfile. Archivo de registro (*log file*). Se usa con el propósito de recuperación.
- \* \$mft. Tabla maestra de archivos (*master file table* o MFT). Contiene un registro para cada uno de los archivos en el volumen (archivo, directorios y *metafile*). Los datos de cada registro incluyen metadatos, tales como: nombre, fecha de creación, permisos de acceso (se usa una lista de control de acceso para esto último) y longitud.

Si el archivo referido es un directorio, la entrada contendrá el nombre y un identificador de archivo, que es el número de registro que representa el archivo en la tabla maestra de archivos.

La tabla MFT soporta algoritmos para minimizar la fragmentación del disco.

- \* \$mftmirr. Tabla maestra de archivos 2 (*master file table 2* o MFT2). Es un espejo de la MFT.

---

31 Un **exabyte** (EB) equivale a  $10^{18}$  bytes, es decir,  $10^6$  TB =  $10^9$  GB.

- \* \$quota. Tabla de cuotas (*quota table*). Indica la cuota de espacio en disco para cada usuario.
- \* \$upcase. Tabla de mayúsculas (*uppercase table*). Se usa para convertir los caracteres en mayúscula, y en minúscula al conjunto de caracteres en mayúscula del código Unicode.
- \* \$volume. Volumen (*volume*). Contiene información del volumen (nombre, versión, etc.).
- \* \$Secure. Archivo de seguridad (*secure file*). Contiene un descriptor único de seguridad para todos los archivos del volumen.
- \* Además, la referencia punto (.) que es un índice al directorio raíz del volumen.

#### Archivos y torrentes (*streams*):

Toda la información acerca de los archivos se almacena en registros MFT: su nombre, longitud, posición en el disco, etc., para lo cual se pueden usar uno o varios registros MFT que no tienen que estar contiguos.

#### Los directorios

Un directorio NTFS es un archivo que especifica referencias a otros archivos y directorios. Está dividido en bloques y cada uno de ellos contiene un nombre de archivo, el atributo base y la referencia al elemento MFT que contiene la información completa del directorio. La estructura interna del directorio es un árbol binario, lo cual permite búsquedas más rápidas.

#### II.8.2 Sistema de la familia Unix

Los SO de la familia Unix pueden usar diferentes sistemas de archivos, entre los que se destacan el sistema de archivo UFS y el extX.

#### Generalidades de los sistemas de archivos Unix

Un archivo Unix es simplemente una secuencia de bytes. Las dos estructuras de datos para localizar un archivo son:

- \* La tabla de directorio.
- \* El nodo-i (*i-node*).

La figura II.13 da una idea general de esta organización. En este caso, se muestra un disco con una tabla de directorio que contiene dos archivos ordinarios (f1 y f2) y un directorio (D1). Cada entrada de la tabla de directorio apunta a un nodo-i que se detallará más adelante, por ahora es importante comprender que existe un nodo-i para cada archivo o directorio.

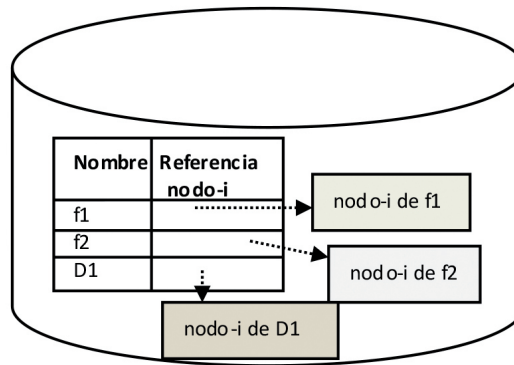


Figura II.13. Relación entre la tabla de directorio y el nodo-i. Fuente elaboración propia.

La tabla de directorio es una estructura de datos que solo contiene dos campos:

- \* El primer campo contiene el nombre del archivo.
- \* El segundo campo contiene un puntero, o referencia, a otra estructura de datos, denominada **nodo-i**.

Aunque D1 es un subdirectorio del directorio que se muestra como ejemplo, no existe mucha diferencia entre él y los archivos f1 y f2, debido a que un subdirectorio no es más que un **archivo especial** que contiene referencias a otros archivos (su contenido es en realidad una tabla de directorio).

Algunos nombres de directorios son estándar en todas las versiones de UNIX (al menos en la mayoría), por ejemplo:

- \* /bin. Contiene los comandos más comunes.
- \* /dev. Contiene manipuladores de dispositivos (*device drivers*) especiales que controlan el acceso a los periféricos.
- \* /etc. Contiene programas y archivos de datos del sistema. Los archivos en el directorio /etc/default contienen información que el sistema usa por defecto.
- \* /lib. Contiene bibliotecas para el lenguaje C y otros lenguajes de programación.
- \* /mnt. Es un directorio vacío reservado para montar sistemas de archivos.
- \* /tmp. Contiene archivos temporales creados por programas del SO. Los archivos están presentes cuando se está ejecutando el programa.
- \* /usr. Aloja los directorios de todos los usuarios del sistema y otros directorios que contienen comandos y archivos de datos adicionales.

En muchos SO, los directorios contienen una gran cantidad de información acerca de los atributos de los archivos. Todo SO necesita de esa información, pero para el caso de los sistemas de archivo que se están analizando, el directorio solo contiene dos

campos (figura II.13); el resto de la información acerca de un archivo dado está dentro del nodo-i.

### El nodo-i (*i-node*) o bloque de índices

El nodo-i es una estructura muy importante dentro del sistema de archivo. Cada archivo posee un nodo-i que se referencia desde la tabla de directorio. Esa estructura de datos posee mucha información acerca del archivo y, además, contiene un conjunto de punteros que permiten encontrar los lugares del disco donde está almacenado el archivo.

La figura II.14 muestra una idea esquemática del nodo-i; los primeros campos contienen información acerca del archivo, por ejemplo, su tamaño y la fecha de creación, entre otras cosas importantes.

Varios campos que contienen información general
Primer puntero directo
...
Último puntero directo
Bloque indirecto
Bloque indirecto doble
Bloque indirecto triple

Figura II.14. El nodo-i. Fuente elaboración propia.

Los punteros a bloques del archivo están representados en la figura II.14 por los campos:

- \* “Primer puntero directo,..., Último puntero directo”. Esos campos (usualmente 12, pero pueden variar entre diferentes implementaciones de Unix) apuntan directamente a bloques de datos desde el mismo nodo-i.
- \* El “Bloque indirecto” se usa cuando un archivo posee más bloques que los que se pueden apuntar directamente desde el nodo-i. En ese caso, este campo apunta a un bloque que solo contiene índices, los cuales apuntan a bloques de datos.
- \* El “Bloque indirecto doble” se usa cuando no son suficientes los dos direccionamientos anteriores. El campo contiene un puntero que apunta a un bloque de índices, que a su vez apunta a otros bloques de índices desde donde se apunta a los datos, produciendo un doble direccionamiento.
- \* El “Bloque indirecto triple” indica una tercera indirección.

La figura II.15 da una idea gráfica de la forma en que se usan los punteros y el nodo-i.

Un archivo puede estar referenciado desde varios directorios, lo único que necesita el SO es que desde los distintos directorios se apunte al mismo nodo-i. Por ese motivo, también existe un campo dentro del nodo-i que es un contador y controla la **cantidad de referencias** que hay hacia un archivo determinado. Cuando se recibe la orden de

borrar un archivo, lo que se hace es restarle uno a ese contador en el nodo-i y quitar la entrada en el directorio correspondiente. El archivo se borrará físicamente cuando el contador sea cero.

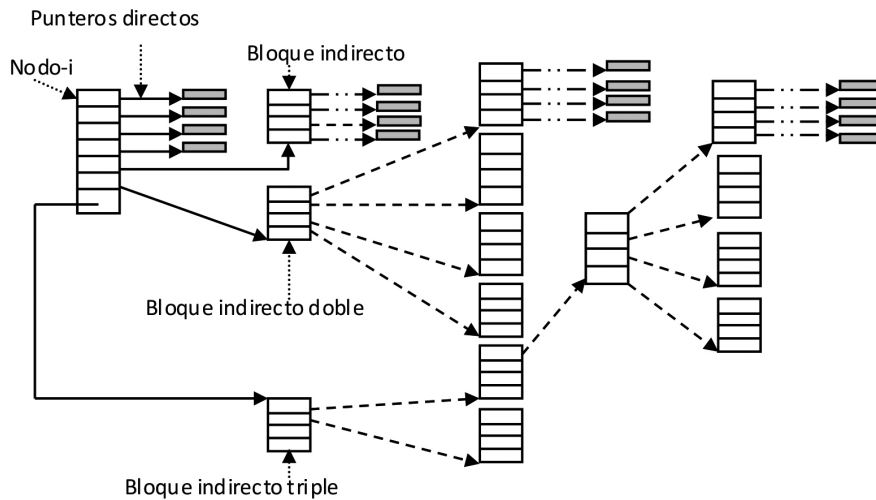


Figura II.15. Los campos apuntables del nodo-i. Los bloques con color son de datos. Fuente elaboración propia.

El SO tiene, al menos, un sistema de archivo que se denomina *root* y se representa por una diagonal (/). Ese sistema de archivo se crea durante el proceso de instalación y contiene todos los programas y directorios del SO, además de otros directorios de usuarios.

A continuación, se presentan dos comandos que se pueden usar en el SO y que ayudan a comprender algunos conceptos (por convenio, lo que se presenta en **negrita** lo escribe el SO).

### Ejemplos de uso de los comandos **df** y **du**

El comando **df** muestra la utilización del equipo de almacenamiento masivo. Si no se le proporciona un nombre, la información se refiere a todos los sistemas de archivos que estén montados<sup>32</sup> en ese momento. En el ejemplo que sigue, se usa el comando **df** con las opciones: **-P** (*paragraph*) para que brinde la información con un cierto formato y **-h** (*human*) para que la información sea más fácil de entender.

```
mlezcano@hercules:~$ df -P -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda1	927M	153M	727M	18%	/
<b>tmpfs</b>	<b>503M</b>	<b>0</b>	<b>503M</b>	<b>0%</b>	<b>/dev/shm</b>
/dev/sdb5	112G	25G	88G	22%	/home
/dev/sda7	927M	8.3M	871M	1%	/tmp
/dev/sda5	9.1G	2.1G	6.5G	25%	/usr
dev/sda8	60G	1.1G	56G	2%	/var

```
mlezcano@hercules:~$
```

<sup>32</sup> Integrar el sistema de archivos de un dispositivo en el árbol de directorio.

Como el comando no especifica ningún nombre, la información se refiere a los sistemas de archivos montados en el equipo. La primera columna contiene sus nombres; la segunda informa sus tamaños; la tercera y la cuarta, los espacios utilizados y los disponibles (respectivamente); la quinta, el porcentaje de uso del sistema de archivo; y la última, el lugar donde está montado cada sistema de archivo, por ejemplo, `/dev/sda1` está montado en el directorio raíz (`/`).

```
mlezcano@hercules:~$ df -i
```

Filesystem	Inodes	IUsed	IFree	IUse%	Mounted on
<code>/dev/sda1</code>	<b>245760</b>	<b>14554</b>	<b>231206</b>	<b>6%</b>	<code>/</code>
<code>tmpfs</code>	<b>128525</b>	<b>1</b>	<b>128524</b>	<b>1%</b>	<code>/dev/shm</code>
<code>/dev/sdb5</code>	<b>0</b>	<b>0</b>	<b>0</b>	<b>-</b>	<code>/home</code>
<code>/dev/sda7</code>	<b>245760</b>	<b>502</b>	<b>245258</b>	<b>1%</b>	<code>/tmp</code>
<code>/dev/sda5</code>	<b>2443200</b>	<b>113653</b>	<b>2329547</b>	<b>5%</b>	<code>/usr</code>
<code>/dev/sda8</code>	<b>16121856</b>	<b>10325</b>	<b>16111531</b>	<b>1%</b>	<code>/var</code>

```
mlezcano@hercules:~$
```

En el ejemplo anterior, se usa el comando `df` para conocer información acerca de los nodos-i de cada sistema de archivo, para lo cual se utiliza la opción `-i`. Las columnas contienen los nombres de los sistemas de archivos y la cantidad de nodos-i (Inodes) de cada uno de ellos, especificando: los nodos-i usados (IUsed), los libres (IFree) y el porcentaje de uso (IUse%); por último, se aprecia el lugar donde están montados los sistemas de archivos (Mounted On).

El comando `du` reporta la cantidad de espacio usado por archivos y directorios; si no se especifica un nombre, el reporte se refiere al directorio actual. Por defecto, el espacio se expresa en unidades de 1024 bytes.

En el siguiente ejemplo, se usa el comando `du` para obtener un reporte del directorio actual.

```
mlezcano@hercules:~$ du
```

<b>4</b>	<code>./mc/cedit</code>
<b>20</b>	<code>./mc</code>
<b>324</b>	<code>./Bash/Lab4</code>
<b>324</b>	<code>./Bash</code>
<b>16</b>	<code>./Perl/C6</code>
<b>24</b>	<code>./Perl/C7</code>
<b>20</b>	<code>./Perl/C8</code>
<b>60</b>	<code>./Perl</code>
<b>548</b>	<code>.</code>

```
mlezcano@hercules:~$
```

### El sistema de archivo UFS (UNIX File System)

El sistema UFS es un descendiente del sistema de archivo original que usó Unix Versión 7 y también se conoce por otras denominaciones: Berkeley Fast File System, BSD Fast File System y FFS.



Un volumen UFS está compuesto por las siguientes partes:

- \* Una cierta cantidad de bloques al comienzo de la partición que se reservan para bloques de inicialización (*boot block*).
- \* Un **superbloque** que contiene un número mágico que identifica el volumen como un sistema de archivo UFS, así como otros números que describen la geometría del sistema de archivo, las estadísticas y los parámetros de afinación de comportamiento.
- \* Una colección de grupos de cilindros. Cada grupo de cilindros tiene los componentes siguientes:
  - Una copia de resguardo (*backup*) del superbloque.
  - Un encabezado con estadísticas, listas de bloques libres, etc., acerca del cilindro (similar a las que contiene el superbloque).
  - Una cantidad de bloques para nodos-i y datos. Los bloques para nodos-i se numeran secuencialmente, los primeros se reservan (por razones históricas) y están seguidos de un nodo-i especial para el directorio raíz.

Las primeras versiones del sistema de archivo Unix se denominaban simplemente FS (File System) e incluían el *boot block*, el superbloque, una cierta cantidad de nodos-i y bloques de datos. Este formato era adecuado para los pequeños discos que manejaban los SO Unix de aquella época, pero con los avances de la tecnología los discos crecieron y el formato original comenzó a provocar movimientos excesivos de los cabezales de lectura-escritura (*thrashing*), lo que llevó a la concepción de los grupos de cilindros (con sus nodos-i y bloques de datos) para reducir este inconveniente<sup>33</sup>.

Actualmente, existen diferentes variantes de UFS implementadas por distintos propietarios de sistemas Unix, tales como: SunOS/Solaris, System V Release 4, HP-UX, Tru64 UNIX, 4.4BSD, BSD Unix systems FreeBSD, NetBSD, OpenBSD y DragonFlyBSD.

### El sistema de archivo extendido “ext”

El SO Linux incluye una implementación de UFS para lograr compatibilidad al nivel de lectura binaria con otros Unix, pero no es una implementación estándar.

Ext (*extended file system*)<sup>34</sup> fue el primer sistema que se creó específicamente para los SO Linux y se inspiró en UFS. Después de un tiempo, ext fue reemplazado por otros dos sistemas: ext2 y xiaf.

### El sistema de archivo extendido “ext2”

Es un sistema de archivos para el *kernel* de Linux. Su principal desventaja es que no implementa el registro por diario (*journaling*) que sí implementa su sucesor ext3.

<sup>33</sup> Ideado por Marshall Kirk McKusick para el 4.2BSD'S FFS (Fast File System).

<sup>34</sup> Fue diseñado por Remy Card para resolver limitaciones de Minix.



Ext2 (*second extended file system*) fue el sistema de archivo por defecto de las distribuciones de Linux Red Hat, Fedora Core y Debian, hasta que fue reemplazado por su sucesor ext3.

El sistema de archivo tiene una tabla de tamaño fijo, donde se almacenan los nodos-i. El tamaño de los bloques se puede especificar cuando se crea el sistema de archivos (desde 512 bytes hasta 4 KB).

### El sistema de archivo extendido “ext3”

El sistema de archivo ext3 (*third extended file system*) soporta el registro de archivos por diario (*journaling*) y se usó extensivamente en las distribuciones Linux. Hoy en día, está siendo sustituido por su sucesor ext4.

La principal diferencia con ext2 es, precisamente, el registro por diario. Los sistemas ext3 utilizan un árbol binario balanceado (AVL)<sup>35</sup> e incorporan el sistema Orlov<sup>36</sup> para asignar los bloques de disco. En los árboles AVL, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa, lo cual permite agilizar la búsqueda.

La velocidad que se logra en los sistemas ext3 es menor que en los sistemas JFS<sup>37</sup>, ReiserFS<sup>38</sup> y XFS; el sistema también es menos escalable que estos últimos, pero se puede actualizar de un sistema de archivos ext2 a uno ext3 sin perder datos, lo cual es una innegable ventaja en relación con los otros sistemas de archivos mencionados en este párrafo.

El sistema de archivos ext3 permite direccionar hasta  $2^{32}$  bloques, que pueden tener diferentes tamaños.

### El sistema de archivo extendido “ext4”

El sistema de archivo ext4 (*fourth extended file system*), también permite el registro por diario o bitácora (*journaling*). Su primera aparición fue de orden experimental; desde el 2008, dejó de ser un experimento que se convirtió en el sustituto de ext3 y es compatible con este último.

El sistema de archivos ext4 es capaz de trabajar con volúmenes de hasta un exabyte y con archivos de hasta 16 TiB.

Introduce el concepto de *extents*, que es distinto al esquema de bloques. Un *extent* es un conjunto de bloques físicos contiguos que se pueden asignar como un todo, lo que mejora el rendimiento cuando se trabaja con archivos grandes al reducir la fragmentación.

En ext4, es posible hacer pre reserva de espacio en disco, para lo cual se usa la llamada al sistema `preallocate()`. El espacio pre reservado para un archivo tiene una buena probabilidad de quedar contiguo.

Otra de las ventajas de ext4 es la asignación retrasada de espacio en disco. Esta técnica retrasa la reservación de bloques del disco hasta que esté cerca el momento de

<sup>35</sup> El nombre del algoritmo AVL se deriva de los apellidos de sus inventores: Georgii Adelson-Velskii y Yevgeniy Landis.

<sup>36</sup> El sistema para asignar bloques Orlov se originó en BSD.

<sup>37</sup> JFS (Journaling File System). Sistema de archivos de 64-bit con registro de diario.

<sup>38</sup> ReiserFS toma el nombre de su autor (Hans Reiser). Sistema de archivos de propósito general con *journaling*.

escribirla (en otros sistemas de archivos la reserva se hace antes), con lo cual se mejora el rendimiento y se reduce la fragmentación.

En este sistema de archivo, el nivel de profundidad de los subdirectorios puede ser 64 000 (en ext3 era 32 000).

Existen muchas otras ventajas de este sistema de archivo que deberán ser consultadas en bibliografías específicas.

## II.9 RESUMEN DEL CAPÍTULO

El sistema de archivo administra los volúmenes de almacenamiento.

La arquitectura del sistema de archivo está estructurada por niveles, desde el más cercano a los equipos en sí, o de bajo nivel, hasta el más cercano a los usuarios; en este último nivel, el sistema hace una abstracción de las características físicas de los equipos para presentar los datos agrupados bajo el concepto de archivo.

Los archivos se organizan en directorios, lo que permite localizarlos más fácilmente, a la vez que proporciona una visión más organizada de los volúmenes.

El sistema de archivo debe tener una forma efectiva de controlar los espacios libres y ocupados del volumen, y en general, se destacan tres técnicas para asignar espacios: contigua, enlazada e indexada.

Los sistemas de archivos varían de un SO a otro; cabe destacar los sistemas NTFS y FAT de los SO de la familia Windows y los sistemas UFS y extX de la familia Unix.

La asignación de espacio en disco en los sistemas indexados (son los más usados hoy en día) provoca fragmentación y por eso muchos sistemas de archivos incluyen utilitarios para desfragmentar los discos, con lo cual se mejora el rendimiento del sistema de archivo.

## II.10 EJERCICIOS

1. El SO Windows implementa dos tipos de sistemas de archivos: FAT y NTFS. El sistema FAT fue el primero que se implementó y aún hoy se usa, aunque no es un sistema de archivo seguro.
  - a. Haga un algoritmo para borrar archivos en el sistema FAT.
  - b. El comando chkdsk permite, entre otras cosas, encontrar los clúster perdidos de un equipo de almacenamiento externo. Los clúster perdidos son aquellos que están referenciados como usados en la FAT (tienen un número diferente de cero), pero no pertenecen a ningún archivo. Haga un algoritmo que permita encontrar los clústeres perdidos y cree archivos con las cadenas de referencias que encuentre.
2. Los sistemas de archivos pueden ser de colocación contigua, enlazada e indexada.
  - a. Defina las estructuras de datos imprescindibles para un sistema de archivos de colocación contigua. Use ideas propias, no debe ser algún sistema real.

- i. Haga un algoritmo para copiar archivos en ese SO.
- b. Defina las estructuras de datos imprescindibles para un sistema de archivos de colocación enlazada. Use ideas propias, no debe ser algún sistema real.
  - i. Haga un algoritmo para mover archivos en ese SO.

Nota: debe observar que el algoritmo es menos complejo cuando el movimiento es dentro del mismo disco.
- c. Defina las estructuras de datos imprescindibles para un sistema de archivos de colocación indexada. Use ideas propias, no debe ser algún sistema real.
  - i. Haga un algoritmo para borrar archivos en ese SO.
- 3. Haga un algoritmo para borrar archivos en el sistema de archivo UFS del SO Unix.
- 4. Busque información adicional acerca de los sistemas de archivos: NTFS y ext4.
  - a. Haga un reporte acerca de las ventajas y desventajas de cada uno de esos sistemas de archivos.
  - b. Haga una valoración crítica de ambos sistemas de archivos.
- 5. Busque los comandos del SO Unix relacionados con el sistema de archivo.
  - a. Haga una tabla, con el formato mostrado a continuación, que le permita buscar ayuda cuando necesite usar alguno de esos comandos.

COMANDOS UNIX		
Comando	Propósito	Ejemplos

- 6. Busque los comandos del SO Windows 10 relacionados con el sistema de archivo.
  - a. Haga una tabla, con el formato mostrado a continuación, que le permita buscar ayuda cuando necesite usar alguno de esos comandos.

COMANDOS WINDOWS 10		
Comando	Propósito	Ejemplos

- 7. Explique la idea que debe seguir un mecanismo de desfragmentación de disco.

## II.11 BIBLIOGRAFÍA CONSULTADA

Abraham, S., Baer- Galvin, P., & Gagne, G. (2013). *Operating system concepts* (9.<sup>a</sup> ed.). Nueva Jersey: Jhon Wiley & Sons.

- Dhamdhere, D. M. (2008). *Operating systems. A concept based aproach*. Nueva York: McGraw-Hill Education.
- Elmasri, R., Carrick, A., Levine, D. (2009). *Operating systems: A spiral approach*. Nueva York: McGraw-Hill.
- Gilly, D. (2003). *Unix in a Nutshell*. En *The Unix CD Bookshelf* (3.<sup>a</sup> ed.). Sebastopol: O'Reilly Media.
- Harvey, M., Deitel, P., & Choffnes, D. (2003). *Operating systems* (3.<sup>a</sup> ed.). Nueva Jersey: Prentice Hall.
- Microsoft Corporation. (2015). *Introducing Windows 10 for IT Professionals. Preview Edition*. Alburquerque: Microsoft Press.
- Peek, J., O'Reilly, T., & Loukides, M. (2003). *UNIX Power Tools*. En *The Unix CD Bookshelf* (3.<sup>a</sup> ed.). Sebastopol: O'Reilly Media.
- Peek, J., Todino G., & Strang, J. (2003). *Learning the Unix Operating System*. En *The Unix CD Bookshelf* (3.<sup>a</sup> ed.). Sebastopol: O'Reilly Media.
- Shotts Jr., W. (2012). *The Linux Command Line: A complete introduction*. San Francisco: No Starch Press.
- Stallings, W. (2014). *Operating systems: Internals and design principles* (8.<sup>a</sup> ed.). Nueva York: Pearson.
- Tanenbaum, A. (2006). *Operating systems: Design and implementation* (3.<sup>a</sup> ed.). Nueva Jersey: Prentice Hall.
- Tanenbaum, A., & Bos, H. (2014). *Modern operating systems* (4.<sup>a</sup> ed.). Nueva York: Pearson.