Lección 10

Sistema de Archivos

- Todas las aplicaciones de computadora requieren almacenar y recuperar información. Mientras un proceso está en ejecución, puede almacenar una cantidad limitada de información dentro de su propio espacio de direcciones. Sin embargo, la capacidad de almacenamiento está restringida por el tamaño del espacio de direcciones virtuales.
- Un segundo problema relacionado con el mantenimiento de la información dentro del espacio de direcciones de un proceso es que cuando el proceso termina, la información se pierde. Para muchas aplicaciones (por ejemplo, una base de datos) la información se debe retener durante semanas, meses o incluso indefinidamente.
- Un tercer problema es que frecuentemente es necesario que varios procesos accedan a (partes de) la información al mismo tiempo. Si tenemos un directorio telefónico en línea almacenado dentro del espacio de direcciones de un solo proceso, sólo ese proceso puede tener acceso al directorio. La manera de resolver este problema es hacer que la información en sí sea independiente de cualquier proceso.
- En consecuencia, tenemos tres requerimientos esenciales para el almacenamiento de información a largo plazo:
    1. Debe ser posible almacenar una cantidad muy grande de información.
    2. La información debe sobrevivir a la terminación del proceso que la utilice.
    3. Múltiples procesos deben ser capaces de acceder a la información concurrentemente.

4.1 ARCHIVOS

- **4.1.1 Nomenclatura de archivos**
- Los archivos son un mecanismo de abstracción. Proporcionan una manera de almacenar información en el disco y leerla después. Esto se debe hacer de tal forma que se proteja al usuario de los detalles acerca de cómo y dónde se almacena la información y cómo funcionan los discos en realidad.
- Probablemente, la característica más importante de cualquier mecanismo de abstracción sea la manera en que los objetos administrados son denominados, por lo que empezaremos nuestro examen de los sistemas de archivos con el tema de la nomenclatura de los archivos. Cuando un proceso crea un archivo le proporciona un nombre. Cuando el proceso termina, el archivo continúa existiendo y puede ser utilizado por otros procesos mediante su nombre.
- Las reglas exactas para denominar archivos varían un poco de un sistema a otro, pero todos los sistemas operativos actuales permiten cadenas de una a ocho letras como nombres de archivos legales.
- Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto, como en *prog.c*. La parte que va después del punto se conoce como la **extensión del archivo** y por lo general indica algo acerca de su naturaleza. Por ejemplo, en MS-DOS, los nombres de archivos son de 1 a 8 caracteres, más una extensión opcional de 1 a 3 caracteres. En UNIX el tamaño de la extensión (si la hay) es a elección del usuario y un archivo puede incluso tener dos o más extensiones, como en *paginainicio.html.zip*, donde *.html* indica una página Web en HTML y *.zip* indica que el archivo se ha comprimido mediante el programa *zip*.

- **4.1.2 Estructura de archivos**
- Los archivos se pueden estructurar en una de varias formas. Tres posibilidades comunes se describen en la figura 4-2. El archivo en la figura 4-2(a) es una secuencia de bytes sin estructura: el sistema operativo no sabe, ni le importa, qué hay en el archivo. Todo lo que ve son bytes. Cualquier significado debe ser impuesto por los programas a nivel usuario. Tanto UNIX como Windows utilizan esta metodología.
- La primera configuración en la estructura se muestra en la figura 4-2(b). En este modelo, un archivo es una secuencia de registros de longitud fija, cada uno con cierta estructura interna. El concepto central para la idea de que un archivo sea una secuencia de registros es la idea de que la operación de lectura devuelva un registro y la operación de escritura sobrescriba o agregue un registro.

- El tercer tipo de estructura de archivo se muestra en la figura 4-2(c). En esta organización, un archivo consiste de un árbol de registros, donde no todos son necesariamente de la misma longitud; cada uno de ellos contiene un campo **llave** en una posición fija dentro del registro. El árbol se ordena con base en el campo llave para permitir una búsqueda rápida por una llave específica.

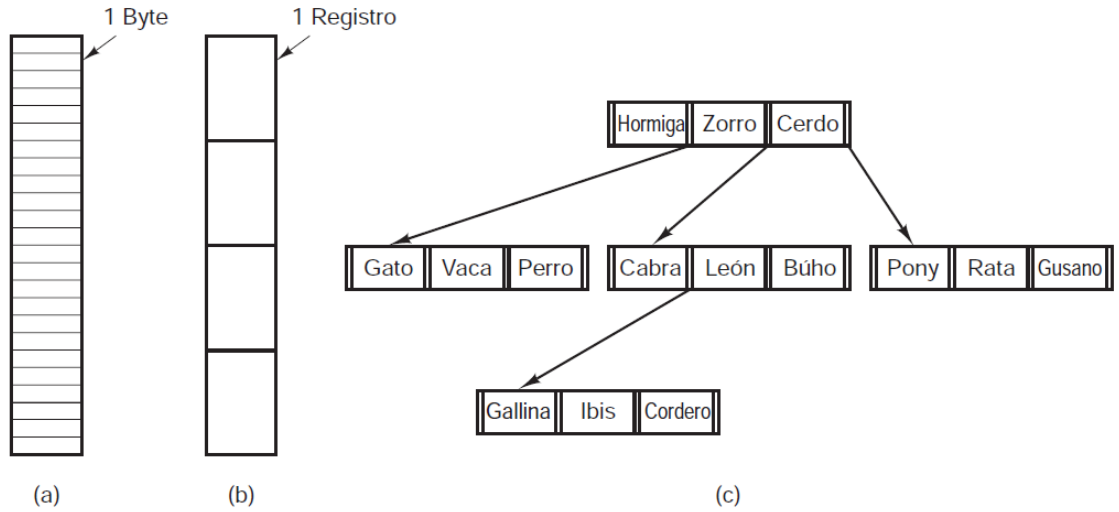

Figura 4-2. Tres tipos de archivos. (a) Secuencia de bytes. (b) Secuencia de registros. (c) Árbol.

- **4.1.3 Tipos de archivos**

- Muchos sistemas operativos soportan varios tipos de archivos. Por ejemplo, UNIX y Windows tienen archivos y directorios regulares. UNIX también tiene archivos especiales de caracteres y de bloques.

- Los **archivos regulares** son los que contienen información del usuario. Todos los archivos de la figura 4-2 son archivos regulares. Los **directorios** son sistemas de archivos para mantener la estructura del sistema de archivos. Estudiaremos los directorios un poco más adelante. Los **archivos especiales de caracteres** se relacionan con la entrada/salida y se utilizan para modelar dispositivos de E/S en serie, tales como terminales, impresoras y redes. Los **archivos especiales de bloques** se utilizan para modelar discos. En este capítulo estaremos interesados principalmente en los archivos regulares.

- Por lo general, los archivos regulares son archivos ASCII o binarios.

- Por ejemplo, en la figura 4-3(a) vemos un archivo binario ejecutable simple tomado de una de las primeras versiones de UNIX. Aunque técnicamente el archivo es sólo una secuencia de bytes, el sistema operativo sólo ejecutará un archivo si tiene el formato apropiado. Este archivo tiene cinco secciones: encabezado, texto, datos, bits de reubicación y tabla de símbolos. El encabezado empieza con un supuesto **número mágico**, que identifica al archivo como un archivo ejecutable (para evitar la ejecución accidental de un archivo que no tenga este formato). Después vienen los tamaños de las diversas partes del archivo, la dirección en la que empieza la ejecución y algunos bits de bandera. Después del encabezado están el texto y los datos del programa en sí. Éstos se cargan en memoria y se reubican usando los bits de reubicación. La tabla de símbolos se utiliza para depurar.

- Nuestro segundo ejemplo de un archivo binario es un archivo, también de UNIX. Consiste en una colección de procedimientos (módulos) de biblioteca compilados, pero no enlazados. A cada uno se le antepone un encabezado que indica su nombre, fecha de creación, propietario, código de protección y tamaño. Al igual que en el caso del archivo ejecutable, los encabezados de los módulos están llenos de números binarios. Al copiarlos a la impresora se produciría basura como salida.
- Cada sistema operativo debe reconocer por lo menos un tipo de archivo —su propio archivo ejecutable— y algunos reconocen más.

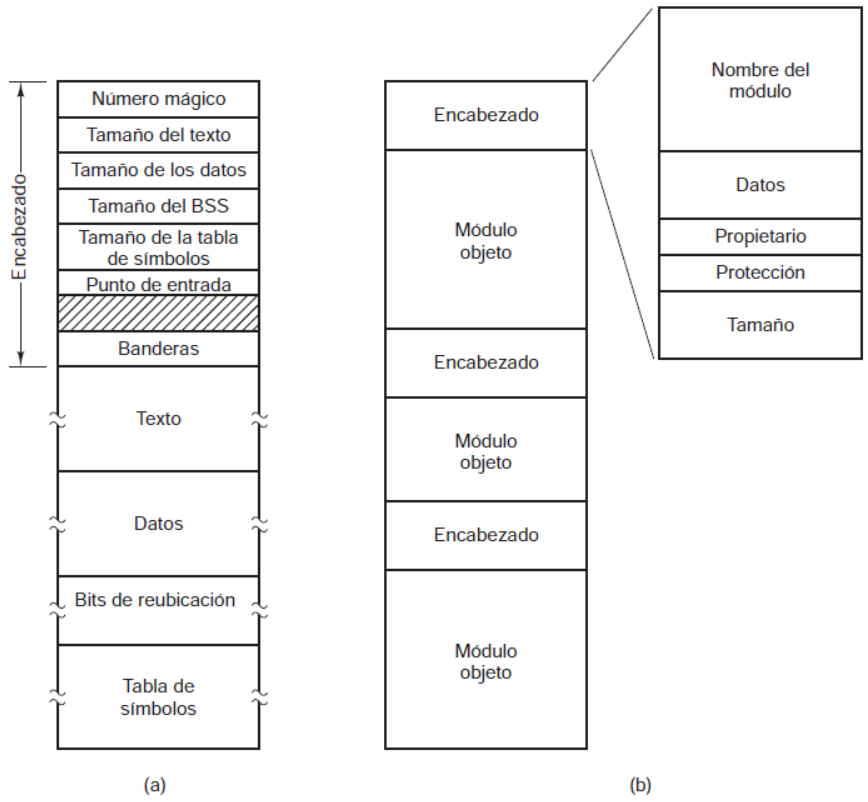

Figura 4-3. (a) Un archivo ejecutable. (b) Un archivo.

- **4.1.4 Acceso a archivos**
- Los primeros sistemas operativos proporcionaban sólo un tipo de acceso: **acceso secuencial**. En estos sistemas, un proceso podía leer todos los bytes o registros en un archivo en orden, empezando desde el principio, pero no podía saltar algunos y leerlos fuera de orden. Sin embargo, los archivos secuenciales podían rebobinarse para poder leerlos todas las veces que fuera necesario. Los archivos secuenciales eran convenientes cuando el medio de almacenamiento era cinta magnética en vez de disco.
- Cuando se empezó a usar discos para almacenar archivos, se hizo posible leer los bytes o registros de un archivo fuera de orden, pudiendo acceder a los registros por llave en vez de posición.
- Los archivos cuyos bytes o registros se pueden leer en cualquier orden se llaman **archivos de acceso aleatorio**. Son requeridos por muchas aplicaciones.
- Los archivos de acceso aleatorio son esenciales para muchas aplicaciones, como los sistemas de bases de datos. Si el cliente de una aerolínea llama y desea reservar un asiento en un vuelo específico, el programa de reservación debe poder tener acceso al registro para ese vuelo sin tener que leer primero los miles de registros de otros vuelos.
- Es posible utilizar dos métodos para especificar dónde se debe empezar a leer. En el primero, cada operación `read` da la posición en el archivo en la que se va a empezar a leer. En el segundo se provee una operación especial (`seek`) para establecer la posición actual. Después de una operación `seek`, el archivo se puede leer de manera secuencial desde la posición actual. Este último método se utiliza en UNIX y Windows.

4.1.5 Atributos de archivos

Todo archivo tiene un nombre y sus datos. Además, todos los sistemas operativos asocian otra información con cada archivo; por ejemplo, la fecha y hora de la última modificación del archivo y su tamaño. A estos elementos adicionales les llamaremos **atributos** del archivo. Algunas personas los llaman **metadatos**. La lista de atributos varía considerablemente de un sistema a otro. La tabla de la figura 4-4 muestra algunas de las posibilidades, pero existen otras.

Atributo	Significado
Protección	Quién puede acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para sólo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivo al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de cero para bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la llave	Desplazamiento de la llave dentro de cada registro
Longitud de la llave	Número de bytes en el campo llave
Hora de creación	Fecha y hora en que se creó el archivo
Hora del último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crecer el archivo

Figura 4-4. Algunos posibles atributos de archivos.

- **4.1.6 Operaciones de archivos**
- Los archivos existen para almacenar información y permitir que se recupere posteriormente. Distintos sistemas proveen diferentes operaciones para permitir el almacenamiento y la recuperación.
- A continuación se muestra un análisis de las llamadas al sistema más comunes relacionadas con los archivos.
  - 1. Create. El archivo se crea sin datos.
  - 2. Delete. Cuando el archivo ya no se necesita, se tiene que eliminar para liberar espacio en el disco.
  - 3. Open. Antes de usar un archivo, un proceso debe abrirlo. El propósito de la llamada a open es permitir que el sistema lleve los atributos y la lista de direcciones de disco a memoria principal para tener un acceso rápido a estos datos en llamadas posteriores.
  - 4. Close. Cuando terminan todos los accesos, los atributos y las direcciones de disco ya no son necesarias, por lo que el archivo se debe cerrar para liberar espacio en la tabla interna.
  - 5. Read. Los datos se leen del archivo. Por lo general, los bytes provienen de la posición actual.
  - 6. Write. Los datos se escriben en el archivo otra vez, por lo general en la posición actual.
  - 7. Append. Esta llamada es una forma restringida de write. Sólo puede agregar datos al final del archivo.
  - 8. Seek. Para los archivos de acceso aleatorio, se necesita un método para especificar de dónde se van a tomar los datos.
  - 9. Get attributes. A menudo, los procesos necesitan leer los atributos de un archivo para realizar su trabajo.
  - 10. Set attributes. Algunos de los atributos puede establecerlos el usuario y se pueden modificar después de haber creado el archivo.
  - 11. Rename. Con frecuencia ocurre que un usuario necesita cambiar el nombre de un archivo existente.

- **4.2 DIRECTORIOS**
- Para llevar el registro de los archivos, los sistemas de archivos por lo general tienen **directorios** o **carpetas**, que en muchos sistemas son también archivos.
- **4.2.1 Sistemas de directorios de un solo nivel**
- La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos. Algunas veces se le llama **directorio raíz**, pero como es el único, el nombre no importa mucho. En las primeras computadoras personales, este sistema era común, en parte debido a que sólo había un usuario. Como dato interesante, la primera supercomputadora del mundo (CDC 6600) también tenía un solo directorio para todos los archivos, incluso cuando era utilizada por muchos usuarios a la vez. Esta decisión sin duda se hizo para mantener simple el diseño del software.
- **4.2.2 Sistemas de directorios jerárquicos**
- Tener un solo nivel es adecuado para aplicaciones dedicadas simples (e incluso se utilizaba en las primeras computadoras personales), pero para los usuarios modernos con miles de archivos, sería imposible encontrar algo si todos los archivos estuvieran en un solo directorio.
- Lo que se necesita es una jerarquía (es decir, un árbol de directorios). Con este esquema, puede haber tantos directorios como se necesite para agrupar los archivos en formas naturales. Además, si varios usuarios comparten un servidor de archivos común, como se da el caso en muchas redes de empresas, cada usuario puede tener un directorio raíz privado para su propia jerarquía.

- **4.2.3 Nombres de rutas**
- Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos métodos distintos. En el primer método, cada archivo recibe un **nombre de ruta absoluto** que consiste en la ruta desde el directorio raíz al archivo. Como ejemplo, la ruta */usr/ast/mailbox* significa que el directorio raíz contiene un subdirectorio llamado *usr*, que a su vez contiene un subdirectorio *ast*, el cual contiene el archivo *mailbox*. Los nombres de ruta absolutos siempre empiezan en el directorio raíz y son únicos.
- El otro tipo de nombre es el **nombre de ruta relativa**. Éste se utiliza en conjunto con el concepto del **directorio de trabajo** (también llamado **directorio actual**). Un usuario puede designar un directorio como el directorio de trabajo actual, en cuyo caso todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo. Por ejemplo, si el directorio de trabajo actual es */usr/ast*, entonces el archivo cuya ruta absoluta sea */usr/ast/mailbox* se puede referenciar simplemente como *mailbox*.

- **4.3 IMPLEMENTACIÓN DE SISTEMAS DE ARCHIVOS**
- Ahora es el momento de cambiar del punto de vista que tiene el usuario acerca del sistema de archivos, al punto de vista del que lo implementa. Los usuarios se preocupan acerca de cómo nombrar los archivos, qué operaciones se permiten en ellos, cuál es la apariencia del árbol de directorios y cuestiones similares de la interfaz. Los implementadores están interesados en la forma en que se almacenan los archivos y directorios, cómo se administra el espacio en el disco y cómo hacer que todo funcione con eficiencia y confiabilidad. En las siguientes secciones examinaremos varias de estas áreas para ver cuáles son los problemas y las concesiones que se deben hacer.
- **4.3.1 Distribución del sistema de archivos**
- Los sistemas de archivos se almacenan en discos. La mayoría de los discos se pueden dividir en una o más particiones, con sistemas de archivos independientes en cada partición. El sector 0 del disco se conoce como el **MBR** (*Master Boot Record*; Registro maestro de arranque) y se utiliza para arrancar la computadora. El final del MBR contiene la tabla de particiones, la cual proporciona las direcciones de inicio y fin de cada partición. Una de las particiones en la tabla se marca como activa.
- Cuando se arranca la computadora, el BIOS lee y ejecuta el MBR. Lo primero que hace el programa MBR es localizar la partición activa, leer su primer bloque, conocido como **bloque de arranque**, y ejecutarlo. El programa en el bloque de arranque carga el sistema operativo contenido en esa partición. Por cuestión de uniformidad, cada partición inicia con un bloque de arranque aunque no contenga un sistema operativo que se pueda arrancar. Además, podría contener uno en el futuro.
- Además de empezar con un bloque de arranque, la distribución de una partición de disco varía mucho de un sistema de archivos a otro. A menudo el sistema de archivos contendrá algunos de los elementos que se muestran en la figura 4-9. El primero es el **superbloque**. Contiene todos los parámetros clave acerca del sistema de archivos y se lee en la memoria cuando se arranca la computadora o se entra en contacto con el sistema de archivos por primera vez. La información típica en el superbloque incluye un número mágico para identificar el tipo del sistema de archivos, el número de bloques que contiene el sistema de archivos y otra información administrativa clave.

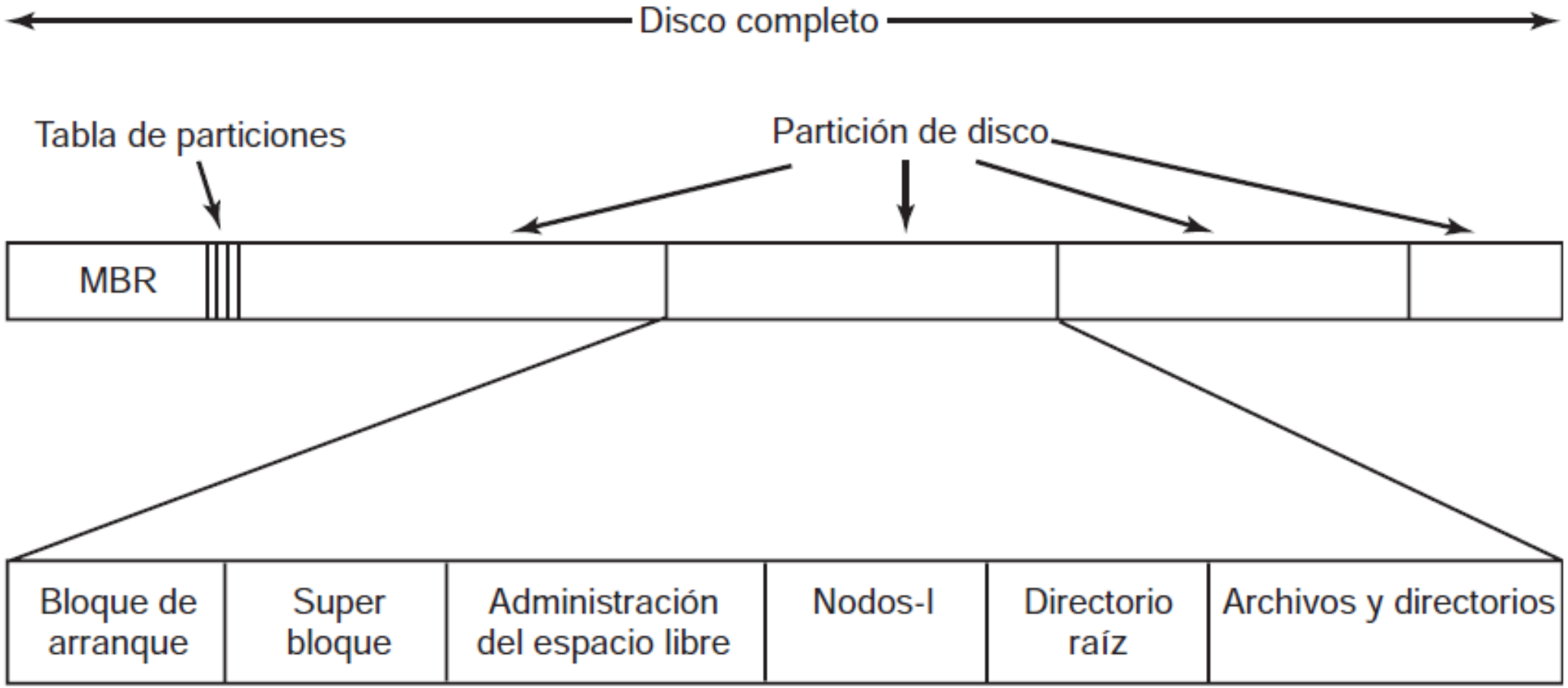

Figura 4-9. Una posible distribución del sistema de archivos.

- **4.3.2 Implementación de archivos**
- Probablemente la cuestión más importante al implementar el almacenamiento de archivos sea mantener un registro acerca de qué bloques de disco van con cuál archivo. Se utilizan varios métodos en distintos sistemas operativos. En esta sección examinaremos unos cuantos de ellos.
- **Asignación contigua**
- El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco. Así, en un disco con bloques de 1 KB, a un archivo de 50 KB se le asignarían 50 bloques consecutivos. Con bloques de 2 KB, se le asignarían 25 bloques consecutivos.

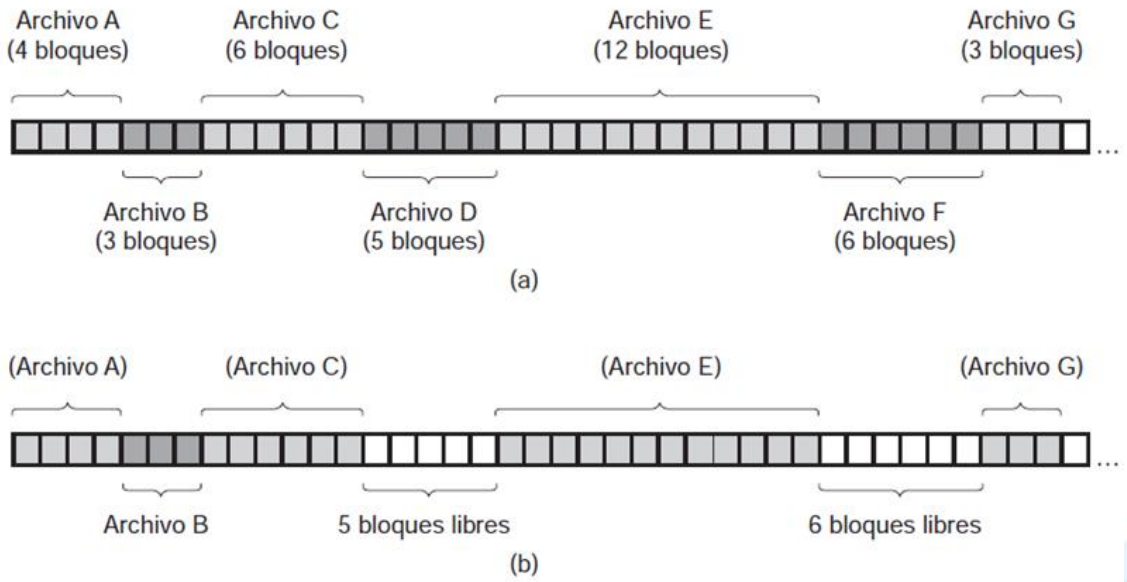

Figura 4-10. (a) Asignación contigua de espacio de disco para siete archivos. (b) El estado del disco después de haber removido los archivos *D* y *F*.

- **Asignación de lista enlazada (ligada)**
- El segundo método para almacenar archivos es mantener cada uno como una lista enlazada de bloques de disco, como se muestra en la figura 4-11. La primera palabra de cada bloque se utiliza como apuntador al siguiente. El resto del bloque es para los datos.

- A diferencia de la asignación contigua, en este método se puede utilizar cada bloque del disco.
- No se pierde espacio debido a la fragmentación del disco (excepto por la fragmentación interna en el último bloque). Además, para la entrada del directorio sólo le basta con almacenar la dirección de disco del primer bloque. El resto se puede encontrar a partir de ella.
- Por otro lado, aunque la lectura secuencial un archivo es directa, el acceso aleatorio es en extremo lento. Para llegar al bloque *n*, el sistema operativo tiene que empezar desde el principio y leer los $n - 1$ bloques anteriores, uno a la vez.

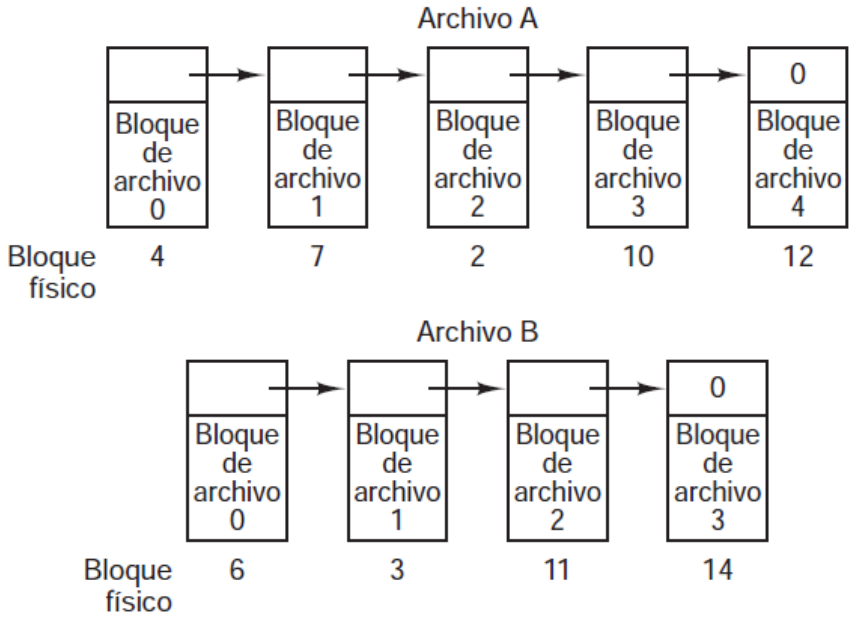

Figura 4-11. Almacenamiento de un archivo como una lista enlazada de bloques de disco.

Asignación de lista enlazada utilizando una tabla en memoria

Las desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del apuntador de cada bloque de disco y la colocamos en una tabla en memoria. La figura 4-12 muestra cuál es la apariencia de la tabla para el ejemplo de la figura 4-11. En ambas figuras tenemos dos archivos. El archivo *A* utiliza los bloques de disco 4, 7, 2, 10 y 12, en ese orden y el archivo *B* utiliza los bloques de disco 6, 3, 11 y 14, en ese orden. Utilizando la tabla de la figura 4-12, podemos empezar con el bloque 4 y seguir toda la cadena hasta el final. Lo mismo se puede hacer empezando con el bloque 6. Ambas cadenas se terminan con un marcador especial (por ejemplo, -1) que no sea un número de bloque válido. Dicha tabla en memoria principal se conoce como **FAT** (*File Allocation Table*, Tabla de asignación de archivos).

FAT no es útil en discos grandes.

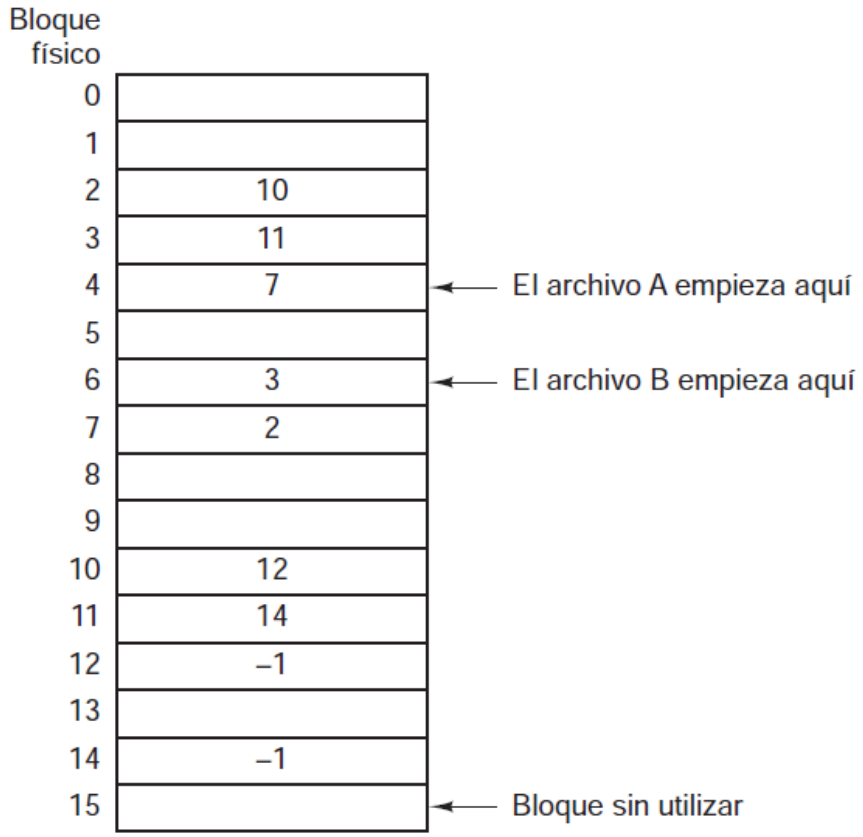

Figura 4-12. Asignación de lista enlazada que utiliza una tabla de asignación de archivos en la memoria principal.

Nodos-i

Asociar a cada archivo una estructura de datos conocida como **nodo-i** (**nodo-índice**), la cual lista los atributos y las direcciones de disco de los bloques del archivo. En la figura 4-13 se muestra un ejemplo simple. Dado el nodo-i, entonces es posible encontrar todos los bloques del archivo. La gran ventaja de este esquema, en comparación con los archivos vinculados que utilizan una tabla en memoria, es que el nodo-i necesita estar en memoria sólo cuando está abierto el archivo correspondiente.

Si cada nodo-i ocupa *n* bytes y puede haber un máximo de *k* archivos abiertos a la vez, la memoria total ocupada por el arreglo que contiene los nodos-i para los archivos abiertos es de sólo *kn* bytes. Sólo hay que reservar este espacio por adelantado.

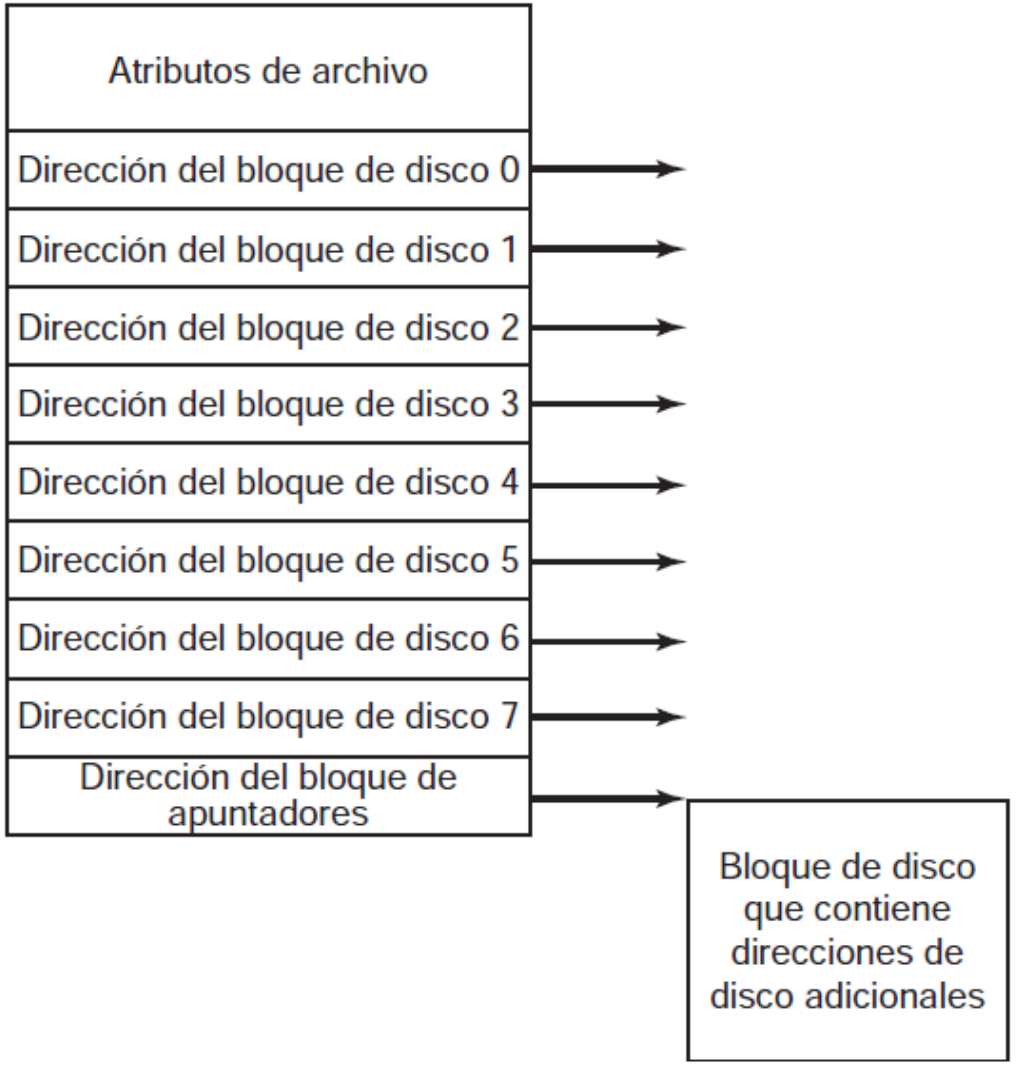

Figura 4-13. Un nodo-i de ejemplo.

- **4.3.3 Implementación de directorios**
- Antes de poder leer un archivo, éste debe abrirse. Cuando se abre un archivo, el sistema operativo utiliza el nombre de la ruta suministrado por el usuario para localizar la entrada de directorio. Esta entrada provee la información necesaria para encontrar los bloques de disco. Dependiendo del sistema, esta información puede ser la dirección de disco de todo el archivo (con asignación contigua), el número del primer bloque (ambos esquemas de lista enlazada) o el número del nodo-i. En todos los casos, la función principal del sistema de directorios es asociar el nombre ASCII del archivo a la información necesaria para localizar los datos.
- Una cuestión muy relacionada es dónde deben almacenarse los atributos. Cada sistema de archivos mantiene atributos de archivo, como el propietario y la hora de creación de cada archivo, debiendo almacenarse en alguna parte. Una posibilidad obvia es almacenarlos directamente en la entrada de directorio. Muchos sistemas hacen eso. Esta opción se muestra en la figura 4-14(a). En este diseño simple, un directorio consiste en una lista de entradas de tamaño fijo, una por archivo, que contienen un nombre de archivo (de longitud fija), una estructura de los atributos del archivo y una o más direcciones de disco (hasta cierto máximo) que indique en dónde se encuentran los bloques de disco.

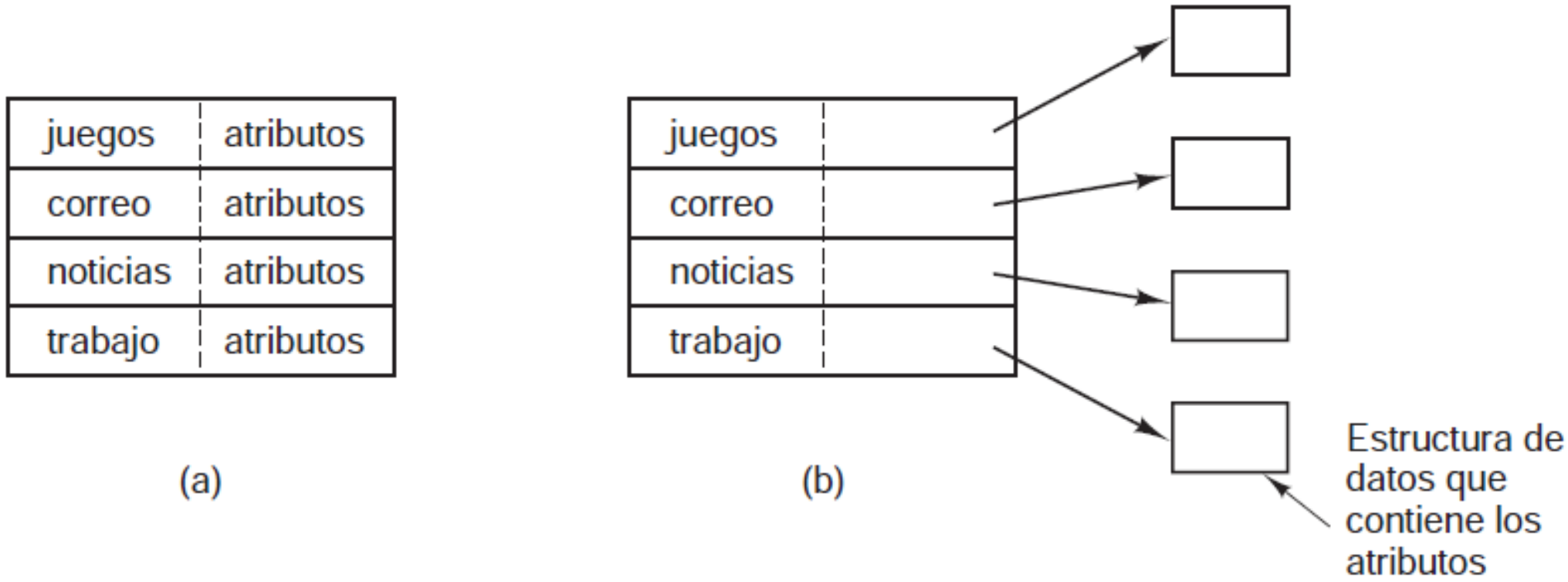

Figura 4-14. (a) Un directorio simple que contiene entradas de tamaño fijo, con las direcciones de disco y los atributos en la entrada de directorio. (b) Un directorio en el que cada entrada sólo hace referencia a un nodo-i.

- **4.4 ADMINISTRACIÓN Y OPTIMIZACIÓN DE SISTEMAS DE ARCHIVOS**
- **4.4.1 Administración del espacio en disco**
- Por lo general los archivos se almacenan en disco, así que la administración del espacio en disco es una cuestión importante para los diseñadores de sistemas de archivos. Hay dos estrategias generales posibles para almacenar un archivo de *n* bytes: se asignan *n* bytes consecutivos de espacio en disco o el archivo se divide en varios bloques (no necesariamente) contiguos. La misma concesión está presente en los sistemas de administración de memoria, entre la segmentación pura y la paginación.
- Como hemos visto, almacenar un archivo como una secuencia contigua de bytes tiene el problema obvio de que si un archivo crece, probablemente tendrá que moverse en el disco. El mismo problema se aplica a los segmentos en memoria, excepto que la operación de mover un segmento en memoria es rápida, en comparación con la operación de mover un archivo de una posición en el disco a otra. Por esta razón, casi todos los sistemas de archivos dividen los archivos en bloques de tamaño fijo que no necesitan ser adyacentes.