Sistemas Operativos I

Módulo V

ENTRADA/SALIDA
y
ARCHIVOS

Temas del Módulo V

- **Introducción.**
  - Conceptos generales.
  - Discos rígidos magnéticos.
- **Gestión de E/S.**
  - Dispositivos de E/S. Aspectos de diseño.
  - Estructura lógica del sistema de E/S.
- **Planificación del disco.**
  - Justificación. FIFO, SSTF, SCAN y C-SCAN.
  - Comparación de rendimientos.
- **Archivos.**
  - Descripción general.
  - Directorios.
  - Diseño del sistema de archivos.
  - Implementación de archivos.
  - Gestión del espacio libre.

Introducción

Conceptos generales.

- La E/S es quizás el aspecto más intrincado en el diseño de un SO, debido a la gran variedad de dispositivos y aplicaciones de éstos.
- Veremos en éste módulo cómo el SO debe gestionar, por un lado los dispositivos de E/S, en particular los de almacenamiento secundario, repasando cómo funciona un disco rígido magnético y sus parámetros de rendimiento.
- Por otro lado, veremos qué es un archivo y un sistema de archivos, de qué manera se organizan y almacenan los archivos, y cómo se gestiona el espacio libre en disco.

Introducción

Discos rígidos magnéticos.

- Son los dispositivos de almacenamiento secundario más populares y económicos. Consisten en uno o varios discos de metal o plástico, a los que se denomina platos, cubiertos con un material magnético.
- Los datos se leen y escriben con un dispositivo llamado cabezal, que se mantiene estacionario mientras el plato gira por debajo.
- El cabezal es, básicamente, una bobina conductora por la que circula electricidad y genera un campo magnético. Los datos en el plato se escriben magnetizando su superficie, emitiendo pulsos magnéticos a través del cabezal.
- La lectura de los datos se realiza “capturando” la magnetización de la superficie del plato con cabezal.

Introducción

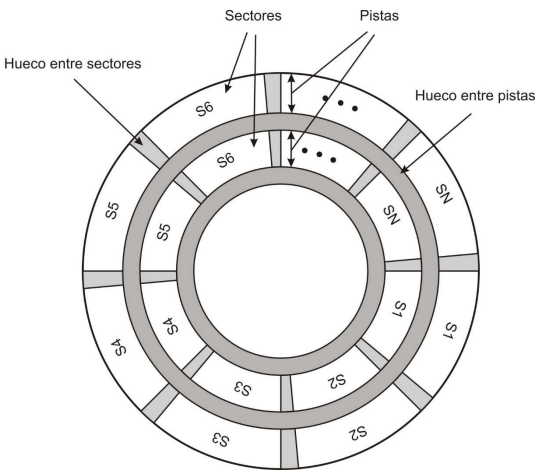
Discos rígidos magnéticos. *(cont.)*

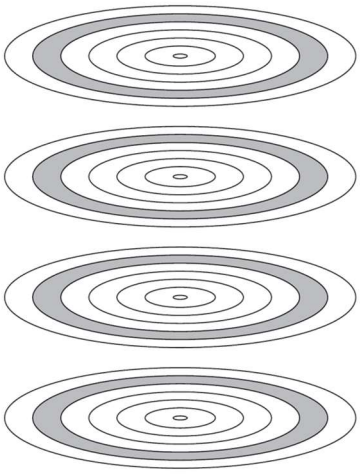
- **Organización y formato de los datos.**
- Los datos en el disco se organizan dividiendo su superficie en:
  - Pistas: es un conjunto de anillos concéntricos, del mismo ancho que el cabezal. Hay miles de pistas en cada superficie. Están separadas por huecos, que sirven para minimizar los errores de lectura/escritura.
  - Sectores: son subdivisiones de una pista. Es la unidad básica de datos, su longitud es normalmente fija, y tiene un tamaño típico de 512 Bytes, siendo éste el tamaño universal de sector. También están separados por huecos.
  - Cilindros: si el disco cuenta con más de un plato, un cilindro está formado por todas las pistas en la misma posición relativa.

Introducción

Discos rígidos magnéticos. *(cont.)*

- **Organización y formato de los datos.** *(cont.)*





Stallings, 2005

Introducción

Discos rígidos magnéticos. *(cont.)*

- **Parámetros de rendimiento.**
- A la hora de la gestionar el disco, el rendimiento del mismo se mide en términos de los siguientes parámetros:
  - Tiempo de búsqueda: es el que tarda el cabezal en posicionarse sobre una pista.
  - Latencia: también llamada *retardo rotacional*, es el tiempo que tarda en llegar el comienzo de un sector por debajo del cabezal. La suma del tiempo de búsqueda y la latencia es el **tiempo de acceso**.
  - Tiempo de transferencia: una vez posicionado el cabezal en la pista, los datos se leen o escriben mientras el sector pasa por debajo, y el tiempo requerido para esa lectura o escritura es el tiempo de transferencia.

Gestión de E/S

Dispositivos de E/S.

- Se pueden clasificar, a grandes rasgos, en tres categorías:
  - Legibles para el usuario: adecuados para la comunicación con el usuario, tales como teclados, monitores, mouse, impresoras, etc.
  - Legibles para la máquina: diseñados para la comunicación con equipamiento electrónico, tales como discos, cintas, sensores, etc.
  - Comunicación: utilizados en la comunicación con equipos remotos, tales como las placas de red, módems, etc.
- Hay grandes diferencias entre las distintas categorías:
  - Velocidad: pueden ser del orden de 10^1 bps como en los teclados, hasta 10^9 bps como en las placas Gigabit Ethernet.
  - Aplicación: un disco, además de almacenar archivos, también se utiliza para implementar esquemas de memoria virtual (Módulo IV). Otro ejemplo es una terminal, que puede ser utilizada por un usuario normal o un administrador.

Gestión de E/S

Dispositivos de E/S. *(cont.)*

- Hay grandes diferencias entre las distintas categorías: *(cont.)*
  - Complejidad de control: una impresora requiere una interfaz relativamente sencilla, mientras que un disco una mucho más compleja.
  - Unidad de transferencia: los datos pueden transferirse como un flujo bytes o caracteres (teclados, pantallas), o bien en bloques de varios bytes (discos).
  - Representación de datos: los dispositivos pueden utilizar distintos esquemas de codificación, por ejemplo, distintos códigos de caracteres (ASCII, UTF-8).
  - Condiciones de error: los tipos de errores, cómo se notifican, etc.
- Las técnicas básicas de comunicación de dispositivos de E/S son tres, y las planteamos en el Módulo Introductorio:
  - E/S programada / Polling.
  - Interrupciones.
  - DMA.

Gestión de E/S

Aspectos de diseño.

- Dos aspectos son muy importantes en el diseño del módulo de E/S de un SO:
  - Eficiencia: los dispositivos de E/S son muy lentos respecto de la memoria y el procesador, por lo que suelen ser un cuello de botella. Una manera de afrontar este problema es la multiprogramación, que como vimos, permite que algunos procesos esperen la finalización de operaciones de E/S mientras que la CPU ejecuta otro proceso.
  - Generalidad: para simplicidad y minimización de errores es deseable manejar todos los dispositivos de manera uniforme. En la práctica esto es muy difícil de lograr por la gran diversidad de características de los dispositivos. Lo que se hace es utilizar una estrategia modular jerárquica para diseñar las funciones de E/S, esto es, modelos de abstracción de varias capas para ocultar las distintas complejidades subyacentes.

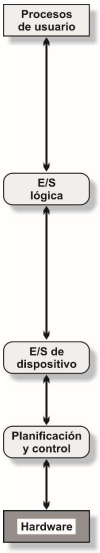
Gestión de E/S

Estructura lógica del sistema de E/S.

- El sistema de E/S se plantea como un modelo jerárquico, y si bien cada fabricante de SO tiene su propio diseño, en términos generales todos contemplan la siguiente organización en tres capas básicas:
  - E/S lógica: es la capa más cercana al usuario. Trata los dispositivos como un recurso lógico sin tener en cuenta los detalles de control real. Gestiona las tareas de E/S para los procesos de usuario en términos de ID de dispositivo, con operaciones sencillas: abrir, cerrar, leer y escribir.
  - E/S de dispositivo: las operaciones y los datos se convierten en instrucciones de E/S, es decir, se encarga de dar las órdenes específicas a los controladores de los dispositivos.
  - Planificación y control: realiza la gestión real de la cola y la planificación de las operaciones de E/S, como también el control de las operaciones. También maneja las IRQ y el estado de la E/S. Esta capa interactúa realmente con el hardware del dispositivo.

Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*



Stallings, 2005

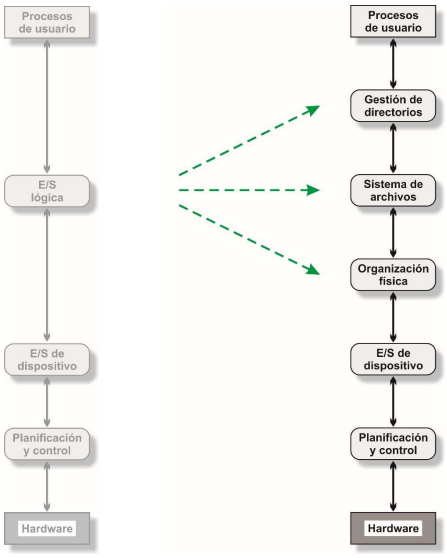
Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*

- En el caso de los dispositivos de almacenamiento secundario que dan soporte a un sistema de archivos, la E/S lógica se divide en:
  - Gestión de directorios: en este nivel se convierten los nombres simbólicos en identificadores de archivos, que pueden referenciarlos directamente, o bien a través de un descriptor o una tabla de índices. También se gestiona las operaciones de usuario, tales como añadir, borrar y reorganizar.
  - Sistema de archivos: trata con la estructura lógica de los archivos y con las operaciones que pueden hacer los usuarios, como abrir, cerrar, leer y escribir. Los derechos de acceso también se gestionan en este nivel.
  - Organización física: se gestiona la traducción de las referencias lógicas a los archivos y registros, en direcciones físicas del almacenamiento secundario, teniendo en cuenta las pistas y los sectores. También se realiza la gestión de la asignación de espacio en disco.

Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*



Stallings, 2005

Planificación del disco

Justificación.

- Como sabemos, los dispositivos de E/S son muy lentos respecto del procesador y la memoria. Por si fuera poco, en las últimas décadas esa brecha de velocidades pasó, aproximadamente, de dos a cuatro órdenes de magnitud.
- Si bien la multiprogramación es un primer paliativo, para mejorar la performance se hace necesario optimizar las operaciones de E/S a fin de minimizar los tiempos de acceso al disco.
- En este sentido, los parámetros de rendimiento del disco vistos son la herramienta fundamental para el diseño de las técnicas de planificación del disco.
- Ejemplos de estas optimizaciones serían minimizar la cantidad de movimientos del cabezal y acceder a los sectores según su cercanía.

Planificación del disco

FIFO (First In-First Out).

- Es la planificación más sencilla, donde se procesan las peticiones de manera secuencial.
- Tiene la ventaja de ser equitativa porque todas las peticiones serán atendidas, siguiendo el orden en que fueron recibidas.
- Si son pocos los accesos a disco, y además las peticiones se corresponden con sectores agrupados de archivos, se puede prever un buen rendimiento.
- Sin embargo, esta técnica tiene en general un rendimiento muy pobre, sobre todo cuando la cola de peticiones es muy grande, fundamentalmente porque no optimiza la cantidad de movimientos del cabezal.

Planificación del disco

SSTF (Shortest Service Time First).

- Consiste en seleccionar la petición de E/S del disco que requiera un menor movimiento del cabezal desde su posición actual. Entonces, siempre se realiza una selección de manera que se produzca un tiempo de búsqueda mínimo.
- Seleccionar siempre el tiempo de búsqueda mínimo no garantiza que sea mínimo el tiempo de búsqueda medio correspondiente a varios movimientos del cabezal. Sin embargo, este esquema proporciona un rendimiento superior al algoritmo FIFO.
- Como el cabezal puede moverse en dos direcciones, se puede utilizar un algoritmo aleatorio para resolver aquellos casos en que la distancia a recorrer hacia ambos lados sea igual.
- Una desventaja importante es que cabe la posibilidad de tener inanición entre las peticiones que requieren más tiempo.

Planificación del disco

SCAN.

- Impide la inanición anterior. Con este algoritmo, el cabezal sólo debe moverse en una dirección, satisfaciendo las peticiones que va encontrando en su camino hasta que llega a la última pista.
- Una mejora, llamada política LOOK, es no llegar hasta la última pista, sino hasta que no haya más peticiones en esa dirección.
- Una vez completadas las peticiones en una dirección, la búsqueda y servicio de las peticiones continúa en la dirección opuesta.
- Este algoritmo se comporta prácticamente igual a SSTF, aún cuando la cola de peticiones cambia dinámicamente.
- La desventaja de este algoritmo es que no aprovecha la proximidad de sectores tanto como SSTF, y puede provocar retardos a las peticiones más recientes.

Planificación del disco

C-SCAN.

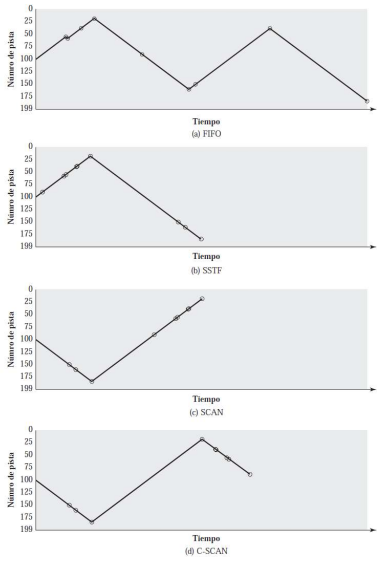
- La política C-SCAN restringe la búsqueda en una sola dirección; después de visitar la última pista, el cabezal vuelve al extremo opuesto del disco y la búsqueda comienza de nuevo.
- Esto reduce el retardo máximo que pueden experimentar las nuevas peticiones. Por otro lado, también reduce el tiempo de espera de las pistas recientemente visitadas.
- Este algoritmo no tiene tan buen rendimiento como SCAN, ya que tiene en promedio más movimientos del cabezal, lo que se puede mejorar utilizando también la política LOOK.

Planificación del disco

Comparación de rendimientos.

Secuencia de pistas solicitadas:
55, 58, 39, 18, 90, 160, 150, 38, 184

Pista inicial: 100



Stallings, 2005

Archivos

Descripción general.

- Todas las aplicaciones requieren almacenar y recuperar información. Mientras un proceso está en ejecución, puede almacenar una cantidad limitada de información dentro de su espacio de direcciones, en memoria principal.
- Un segundo problema es que al almacenar información dentro de su espacio de direcciones, cuando el proceso termina, ésta se pierde, lo que es inaceptable en la gran mayoría de las aplicaciones, como es el caso de las bases de datos.
- Un tercer problema es que suele ser necesario que varios procesos accedan a (partes de) la información al mismo tiempo; esto se resuelve haciendo que la información en sí sea independiente de cualquier proceso.

Archivos

Descripción general. *(cont.)*

- En consecuencia, tenemos tres requerimientos esenciales para el almacenamiento de información a largo plazo:
  - Debe ser posible almacenar una cantidad muy grande de información.
  - La información debe sobrevivir a la terminación del proceso que la utilice.
  - Múltiples procesos deben ser capaces de acceder a la información concurrentemente.
- Así como el SO provee las abstracciones de proceso y memoria virtual, también presenta una abstracción para el almacenamiento: los **archivos**.
- Los archivos son unidades lógicas de información creada por los procesos. La información que se almacena debe ser persistente, es decir, no debe ser afectada por la creación y terminación de los procesos. La parte del SO que los administra se conoce como **sistema de archivos**.

Archivos

Descripción general. *(cont.)*

- Los archivos son un mecanismo de abstracción para proteger al usuario de los detalles acerca de cómo y dónde se almacena y lee la información, y de cómo funcionan los discos en realidad.
- En esta sección, primero analizaremos los archivos desde el punto de vista del usuario, esto es, cómo se utilizan y que propiedades tienen.
- **Nomenclatura de archivos.**
- Cuando un proceso crea un archivo, le otorga un nombre.
- Las reglas exactas para denominar archivos varían de un sistema a otro, pero todos los SO actuales permiten cadenas de una a ocho letras como nombres de archivos legales.
- Algunos sistemas distinguen mayúsculas de minúsculas, y otros no; ejemplos de los primeros, típicamente los sistemas compatibles con POSIX, y de los segundos, MS-DOS, Windows, etc.

Archivos

Descripción general. *(cont.)*

- **Nomenclatura de archivos.** *(cont.)*
- Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto. La parte que va después del punto se conoce como la extensión del archivo y por lo general indica algo acerca de su naturaleza.
- En algunos sistemas, como los basados en POSIX, la extensión es sólo una convención y no son impuestas por el SO. Sin embargo, las aplicaciones pueden requerirlas, como el compilador *gcc* exige que el código fuente tenga la extensión *.c* para programas en C.
- Por el contrario, Windows está consciente de las extensiones y les da un significado. Lo usuarios/procesos pueden registrar extensiones en el SO y especificar qué aplicación “posee” esa extensión.

Archivos

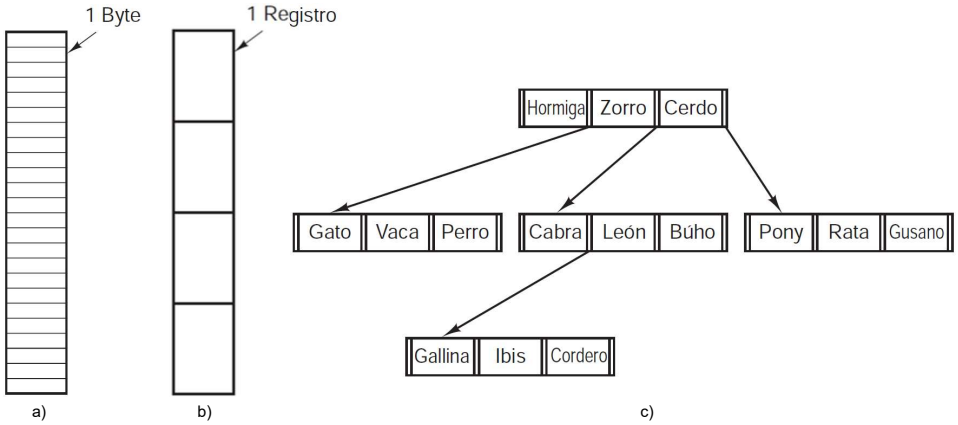
Descripción general. *(cont.)*

- **Estructura de archivos.**
- Existen básicamente tres maneras de estructurar los archivos:
  - Secuencia de bytes: en sí, no hay estructura, el SO 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 POSIX como Windows utilizan esta metodología.
  - Secuencia de registros: en este modelo se tiene una secuencia de registros de longitud fija, cada uno con una cierta estructura interna. Este modelo dejó de utilizarse en sistemas de propósito general.
  - Árbol: en esta estructura, el archivo es un árbol de registros, no necesariamente todos de la misma longitud. Cada registro contiene una *clave*, la cual se utiliza para ordenar el archivo y permitir una búsqueda rápida según este campo. Este tipo de estructura es muy utilizada en los SO de tipo mainframe.

Archivos

Descripción general. *(cont.)*

- **Estructura de archivos.** *(cont.)*



a) Secuencia de bytes. b) Secuencia de registros. c) Árbol.

Tanenbaum, 2009

Archivos

Descripción general. *(cont.)*

- **Acceso a archivos.**
- Hay dos modos elementales para acceder a los archivos:
  - Acceso secuencial: es el más antiguo; los bytes o registros son leídos en orden, empezando desde el principio, y no es posible saltear o leer fuera de orden.
  - Acceso aleatorio: es posible leer los bytes o registros fuera de orden, cualquiera sea la posición del byte o registro.
- En la actualidad los archivos se pueden acceder combinando los dos tipos de acceso.
- Una operación especifica la posición donde se debe empezar a leer, y otra operación establece al posición actual dentro del archivo, para luego realizar lecturas secuenciales.
- Estos métodos son utilizados tanto en POSIX como en Windows.

Archivos

Descripción general. *(cont.)*

- **Atributos de archivos.**
- Todo archivo tiene un nombre y sus datos. Además, todos los SO asocian otra información con cada archivo. A estos elementos adicionales les llamaremos atributos del archivo.
- Ejemplos de atributos de un archivo son:
  - Protección: quién puede acceder al archivo y de qué forma, contraseña, creador, propietario.
  - Banderas: sólo lectura, oculto, del sistema, ASCII/binario, etc.
  - Longitudes: cantidad de bytes por registro, del campo clave.
  - Fechas y horas: de creación, último acceso, última modificación.
  - Tamaños: actual, máximo.

Archivos

Descripción general. *(cont.)*

- **Operaciones de archivos.**
- Las operaciones de archivos están relacionadas con las system calls; las más comunes implementadas por la mayoría de los SO son:
  - Create: crea un archivo sin datos.
  - Delete: elimina el archivo y libera espacio en el almacenamiento.
  - Open: abre un archivo, llevando a memoria los atributos y la lista de direcciones.
  - Close: cierra el archivo, eliminando sus datos de acceso de la tabla en memoria.
  - Read: lee datos, especificando la cantidad de datos a leer y desde qué posición.
  - Write: escribe datos, especificando desde qué posición.
  - Append: similar a write, pero escribe los datos siempre al final del archivo.
  - Seek: posiciona el puntero al archivo, respecto del inicio, final o la posición actual.
  - Get attributes: lee los atributos de un archivo.
  - Set attributes: modifica los atributos de un archivo.
  - Rename: cambia el nombre del archivo.

Archivos

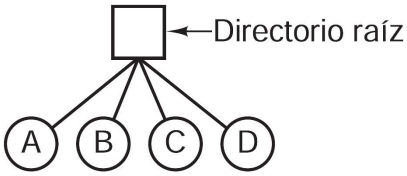
Directorios.

- Para llevar el registro de los archivos, los sistemas de archivos por lo general tienen directorios o carpetas.
- En muchos sistemas operativos, típicamente en aquellos basados en POSIX, los directorios son tratados también como archivos, y al igual que éstos, tienen operaciones implementadas como llamadas al sistema.
- En esta sección hablaremos sobre los directorios, su organización, sus propiedades y las operaciones que pueden realizarse con ellos.

Archivos

Directorios. *(cont.)*

- **Directorios de un solo nivel.**
- La forma más simple de un sistema de directorios es tener un único directorio que contenga todos los archivos.
- En las primeras computadoras personales, este sistema era común, en parte debido a que sólo había un usuario.
- Las ventajas de este esquema son su simpleza y la rapidez para localizar archivos. Se utiliza típicamente en sistemas embebidos.

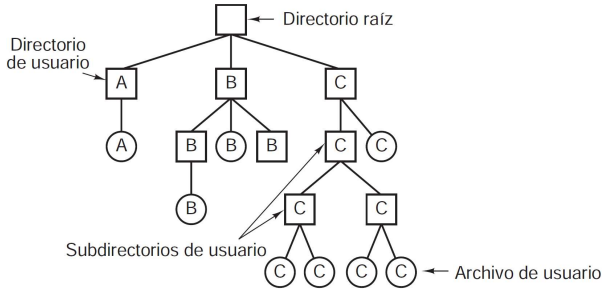


Tanenbaum, 2009

Archivos

Directorios. *(cont.)*

- **Directorios jerárquicos.**
- 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, cada usuario puede tener un directorio raíz privado con su propia jerarquía. Casi todos los sistemas modernos se organizan de esta manera.



Tanenbaum, 2009

Archivos

Directorios. *(cont.)*

- **Nombre 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.
- Por 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. En POSIX, los componentes de la ruta van separados por /; en Windows el separador es \.

Archivos

Directorios. *(cont.)*

- **Nombre de rutas.** *(cont.)*
- 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 designa un directorio como el directorio de trabajo actual, y 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*.

Archivos

Directorios. *(cont.)*

- **Operaciones de directorios.**
- Las operaciones de directorio difieren mucho más que las de archivos de un SO a otro; a modo de ejemplo, veremos algunas en POSIX:
  - Create: crea un directorio vacío.
  - Delete: elimina el directorio; sólo se puede eliminar un directorio vacío.
  - Opendir: abre un directorio, por ejemplo, para listar su contenido.
  - Closedir: cierra el directorio, y libera el espacio en la tabla interna.
  - Readdir: devuelve la siguiente entrada en un directorio abierto.
  - Rename: cambia el nombre del directorio.
  - Link: permite que un mismo archivo aparezca en más de un directorio.
  - Unlink: elimina una entrada en un directorio; si está presente en más de uno, solo se elimina del directorio actual.

Archivos

Diseño del sistema de archivos.

- Cambiamos ahora el punto de vista del usuario sobre el sistema de archivos, al punto de vista del que lo implementa.
- Los sistemas de archivos se almacenan en discos. Éstos se pueden dividir en una o más particiones, cada una con sistemas de archivos independientes.
- El sector 0 del disco se denomina 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.

Archivos

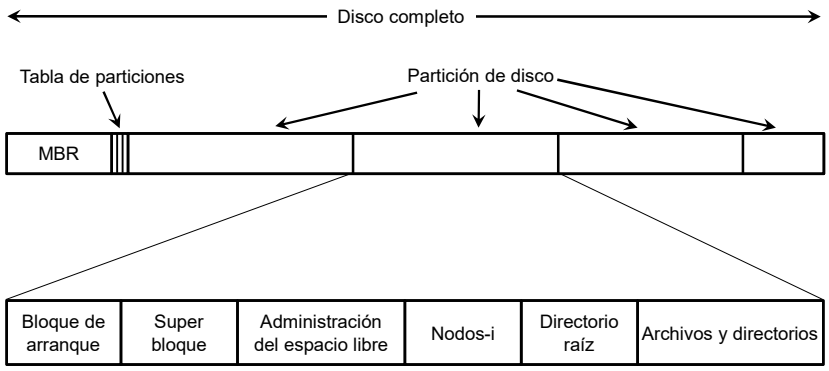
Diseño del sistema de archivos. *(cont.)*

- 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.
- La distribución de una partición de disco varía mucho de un sistema de archivos a otro. A menudo contendrá algunos de estos elementos:
  - Superbloque: se lleva a memoria cuando la computadora inicia, y contiene información para identificar el tipo de sistema de archivo, el número de bloques que contiene, etc.
  - Administración del espacio libre: información acerca del espacio libre en el sistema de archivos, con alguna de las técnicas que veremos más adelante.
  - Nodos-i: arreglo de estructuras de datos, uno por archivo, con la información completa del archivo.

Archivos

Diseño del sistema de archivos. *(cont.)*

- Después de esto, podría venir el directorio raíz, y por último, el resto del disco contiene todos los otros directorios y archivos.



Tanenbaum, 2009

Archivos

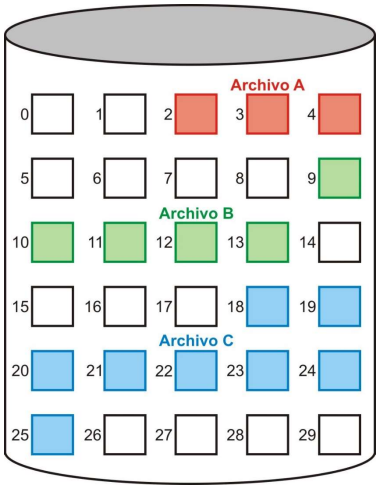
Implementación de archivos.

- Tal vez 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 SO.
- **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.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*



Entradas en el directorio

Nombre de archivo	Bloque inicial	Longitud
Archivo A	2	3
Archivo B	9	5
Archivo C	18	8

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*
- Este método tiene dos ventajas significativas:
- En primer lugar es simple de implementar, ya que llevar un registro de la ubicación de los bloques de un archivo se reduce a recordar dos números: la dirección de disco del primer bloque y el número de bloques en el archivo. Dado el número del primer bloque, se puede encontrar el número de cualquier otro bloque con una simple suma.
- En segundo lugar, el rendimiento de lectura es excelente; el archivo completo se puede leer del disco en una sola operación. Sólo se necesita una búsqueda (para el primer bloque); después de eso, no son necesarias más búsquedas ni retrasos por rotación. Por ende, la asignación contigua tiene un alto rendimiento.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*
- La desventaja de este método es ligeramente significante: con el transcurso del tiempo, los discos se fragmentan, tal como vimos que pasa con la memoria principal.
- Cuando se quita un archivo, sus bloques se liberan naturalmente, dejando una serie de bloques libres en el disco.
- El disco no se compacta al momento para quitar el hueco, ya que eso implicaría tener que copiar todos los bloques que van después del hueco, que podrían ser millones.
- Como resultado, el disco al final consiste de archivos y huecos, difícil de mantener, por lo que esta forma de asignación se dejó de utilizar en discos magnéticos, pero es utilizada en discos ópticos (CD, DVD).

Archivos

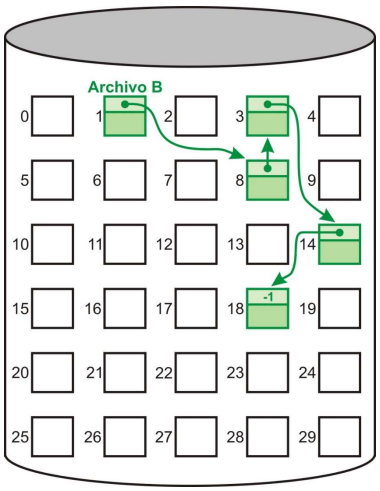
Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.**
- Con este método, un archivo es una lista enlazada de bloques en el disco. La primera palabra de cada bloque se utiliza como puntero al siguiente; el resto del bloque es para los datos. En el último bloque, la primera palabra contendrá un valor numérico que indicará que es el último (cero o negativo).
- Con 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).
- Para la entrada del directorio se almacena la dirección de disco del primer bloque; el resto se puede encontrar a partir de ella.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.** *(cont.)*



Entradas en el directorio

Nombre de archivo	Bloque inicial
…	…
Archivo B	1
…	…

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.** *(cont.)*
- Si bien la lectura secuencial de un archivo es directa, el acceso aleatorio es en extremo lento. Para llegar al bloque *n*, el SO tiene que empezar desde el principio y leer los *n* – 1 bloques anteriores, uno a la vez.
- Por otro lado, la cantidad de datos en un bloque ya no es una potencia de dos, debido a que el puntero ocupa unos cuantos bytes. Aunque no es fatal, tener un tamaño peculiar es menos eficiente debido a que muchos programas leen y escriben en bloques, cuyo tamaño es una potencia de dos.
- Además, no hay principio de proximidad de sectores, lo que provoca que leer varios bloques cuesta acceder a diferentes partes del disco, lo que resulta en una operación de lectura de muy bajo rendimiento.

Archivos

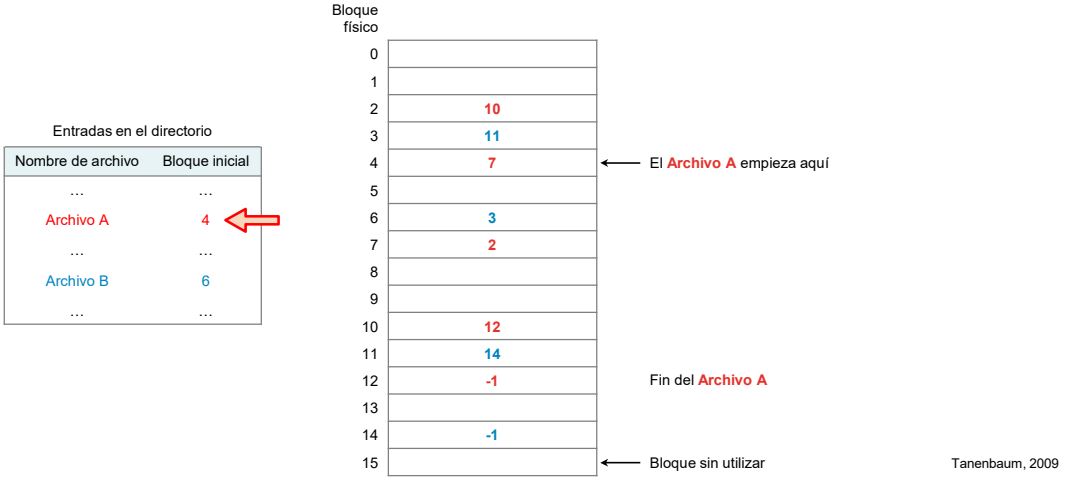
Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.**
- Ambas desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del puntero de cada bloque de disco y la colocamos en una tabla en memoria principal.
- En la entrada del directorio se almacena la dirección del primer bloque, que coincide con el número de entrada en la tabla, y cada entrada en la tabla contiene la dirección del próximo bloque.
- El último bloque se indica con un valor especial en esa entrada de la tabla, que será un número de bloque no válido, por ejemplo, -1.
- De esta manera, un archivo se recorre siguiendo los valores de cada entrada de la tabla, hasta encontrar el indicador de fin del archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*



Tanenbaum, 2009

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*
- Esta tabla en memoria principal se conoce como FAT (File Allocation Table, tabla de asignación de archivos). Con esta organización, el bloque completo está disponible para los datos.
- El acceso aleatorio es mucho más sencillo, y si bien aún se tiene que seguir la lista, al estar en memoria es mucho más rápido al no tener que hacer las sucesivas referencias a disco.
- Al igual que el método anterior, la entrada de directorio necesita sólo un entero (el número de bloque inicial) y aún así se puede localizar todos los bloques, sin importar qué tan grande sea el archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*
- La principal desventaja de este método es que es necesario que la tabla completa esté en memoria todo el tiempo para que funcione.
- Con un disco de 200 GB y un tamaño de bloque de 1 KB, la tabla necesita 200 millones de entradas, una para cada uno de los 200 millones de bloques de disco.
- Con 4 bytes por entrada, el tamaño ocupado por la tabla ronda los 800 MB de memoria, lo cual no es tan práctico.
- Esto hace inviable esta metodología para discos grandes.

Archivos

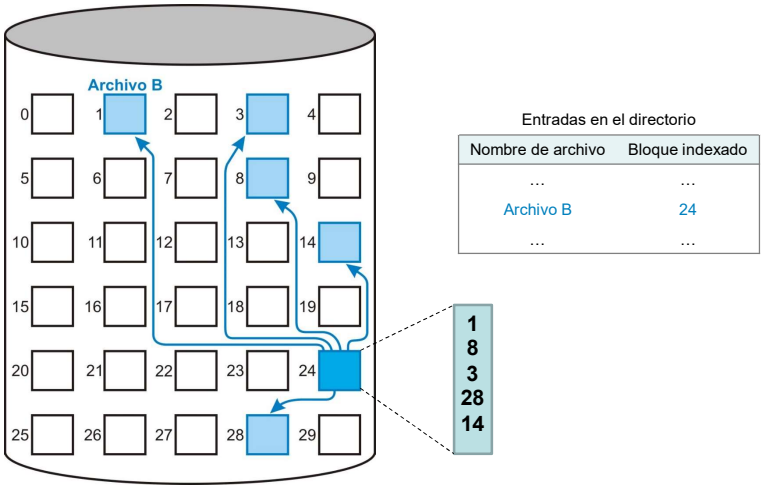
Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).**
- Cada archivo tiene asociada una estructura de datos, llamada nodo-i (nodo-índice), la cual lista los atributos y las direcciones de disco de los bloques del archivo.
- Esta estructura se almacena en el primer bloque del archivo, por lo que la entrada del directorio sólo necesita ésta dirección.
- Los bloques del archivo se pueden acceder entonces a partir de las direcciones almacenadas en el nodo-i.
- La gran ventaja de este esquema, en comparación con los archivos enlazados que utilizan una tabla en memoria, es que el nodo-i sólo necesita estar en memoria cuando está abierto el archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*



Archivos

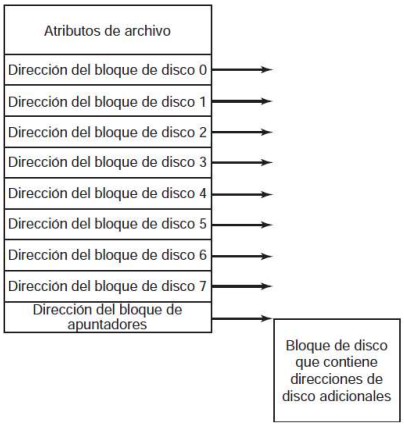
Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*
- Un problema con los nodos-i es que éstos tienen un número fijo de direcciones de disco (limitado por el tamaño del bloque), por lo tanto, el archivo tendrá una cantidad acotada de bloques.
- Si el archivo tiene que crecer más allá de esa cantidad de bloques, la solución es reservar la última dirección, no para un bloque de datos, sino para un bloque que contiene más direcciones de disco.
- A su vez, esos bloques pueden apuntar a otros bloques con más direcciones, permitiendo que el archivo pueda tener tamaños más grandes.
- Esta metodología es ampliamente utilizada en los sistemas POSIX.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*



Tanenbaum, 2009

Archivos

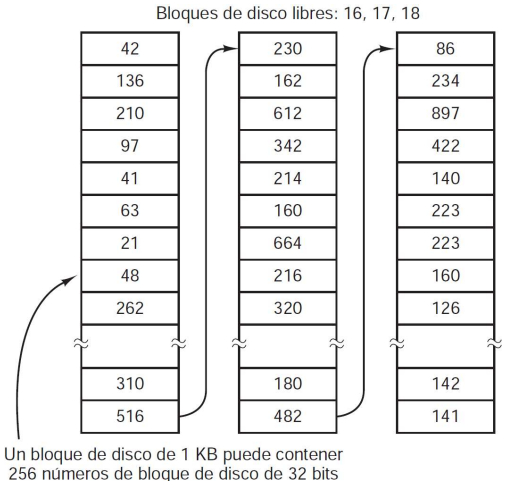
Gestión del espacio libre.

- Hay dos métodos ampliamente utilizados para la gestión del espacio libre en disco: lista enlazada de bloques libres, y mapa de bits.
- **Lista enlazada de bloques libres.**
- Cada bloque contiene tantos números de bloque de disco libres como pueda, dejando libre una palabra dentro del bloque para un puntero al siguiente bloque.
- Con un bloque de 1 KB y un número de bloque de disco de 32 bits, cada bloque en la lista contiene los números de 255 bloques libres.
- Con un disco de 500 GB, unos 488 millones de bloques, se requiere una cantidad aproximada de 1.9 millones de bloques.
- En general se utilizan bloques libres para mantener esta lista, por lo que en esencia su almacenamiento es gratuito.

Archivos

Gestión del espacio libre. *(cont.)*

- **Lista enlazada de bloques libres.** *(cont.)*



Tanenbaum, 2009

Archivos

Gestión del espacio libre. *(cont.)*

- **Lista enlazada de bloques libres.** *(cont.)*
- Este método requiere un solo bloque de punteros en memoria. Al crear un archivo, los bloques necesarios se toman de este bloque de punteros. Cuando se agotan, se lee del disco un nuevo bloque de punteros.
- Una desventaja es que, en ciertas circunstancias, se pueden producir operaciones de E/S innecesarias. Por ejemplo, cuando el bloque de punteros tiene dos direcciones disponibles y se elimina un archivo de tres bloques.
- Entonces el bloque de punteros se desborda, se escribe en disco y se lee uno a memoria. Si inmediatamente se crea un archivo nuevo de tres bloques, se vuelve a la situación anterior, y si ese archivo es temporal, se repiten nuevamente todas las operaciones de E/S.

Archivos

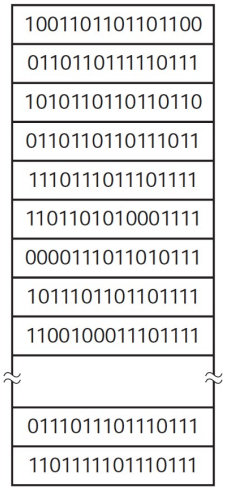
Gestión del espacio libre. *(cont.)*

- **Mapa de bits.**
- Un disco de *n* bloques requiere un mapa de *n* bits. Los bloques libres se representan con un 1, y los ocupados con un 0 (o viceversa).
- Con el disco de 500 GB de ejemplo, necesitamos 488 millones de bits para el mapa, que requieren poco menos de 60.000 bloques, apenas un poco más del 3% requerido por la lista enlazada.
- Con un mapa de bits también es posible mantener sólo un bloque en memoria, usando el disco para obtener otro sólo cuando el primero se llena o se vacía.

Archivos

Gestión del espacio libre. *(cont.)*

- **Mapa de bits.** *(cont.)*



Tanenbaum, 2009

Archivos

Gestión del espacio libre. *(cont.)*

- **Mapa de bits.** *(cont.)*
- Otra ventaja de este método es que al realizar toda la asignación de un solo bloque del mapa de bits, los bloques de disco estarán cerca uno del otro, con lo cual se minimiza el movimiento del cabezal.
- Como el mapa de bits es una estructura de datos de tamaño fijo, si el kernel está paginado, el mapa de bits puede colocarse en memoria virtual y hacer que se paginen sus páginas según se requiera.
- Esto soluciona una posible desventaja, que es que ocupe más lugar en memoria de lo que realmente es necesario.

Fin del Módulo V