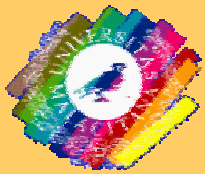


Departamento de Lenguajes y Ciencias de la Computación Universidad de Málaga

Sistemas Operativos

Tema 1:

Sistemas de Ficheros

Curso 2002/2003

E.T.I. Informática Gestión

Profesor:
Francisco López Valverde

Índice de contenidos

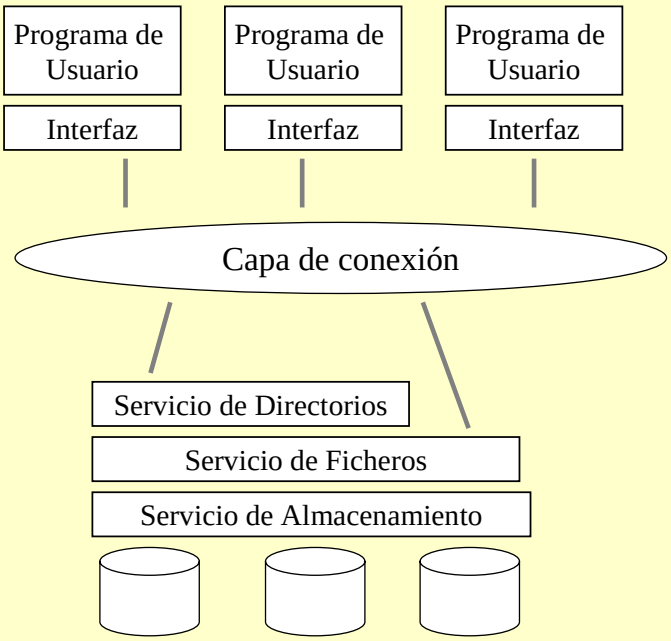
Qué es un sistema de ficheros

- Un sistema de ficheros es aquél componente del sistema operativo responsable de los siguientes aspectos relacionados con los ficheros:
  - Organización (secuencias de bytes, registros, ...)
  - Almacenamiento (contiguo, enlazado, ...)
  - Recuperación (tablas de ficheros, caches, ...)
  - Denominación (nombres de los ficheros)
  - Compartición (enlaces)
  - Protección (mecanismos de protección)

Qué es un sistema de ficheros

- Desde un punto de vista de alto nivel
  - El sistema de ficheros ofrece a los usuarios una interfaz de programación que caracteriza a la abstracción de fichero
  - Se definen los servicios que ofrecen (servicio de ficheros, servicio de directorios, ...)
  - Interesa saber el QUÉ
- Desde un punto de vista de bajo nivel
  - Se trata de cómo implementar los servicios que ofrece el sistema de ficheros
  - Se usa del concepto de servidor como entidad que implementa y ofrece uno o varios servicios
  - Interesa saber el CÓMO

Estructura de un sistema de ficheros



Estructura de un sistema de ficheros

- La capa de conexión puede ser
  - Memoria compartida
    - En los sistemas operativos tradicionales
  - Una red de alta velocidad
    - En los sistemas operativos en red
- Tradicionalmente
  - Los servicios de directorios y ficheros están en el núcleo. En la actualidad no siempre es así
- En la mayoría de los casos
  - Los directorios se implementan como ficheros
  - Aunque algunos sistemas los implementan de forma independiente

Ficheros

- El fichero es un recurso del sistema operativo que ofrece tres funciones básicas:
  - Posibilidad de almacenar grandes volúmenes de información
  - Almacenar la información en dispositivos persistentes (discos, cintas, cdrom, ...)
  - Compartir información entre usuarios
- Es un recurso lógico
  - El sistema operativo proporciona a los usuarios un unidad de almacenamiento lógica, ocultando las propiedades físicas de los dispositivos de almacenamiento

Concepto de fichero

- Para los usuarios
  - Un fichero es una colección de información que se encuentra en un dispositivo de almacenamiento secundario
  - Cada fichero tiene un nombre, en forma de cadena de caracteres
- Para los programadores
  - Un fichero es un tipo abstractos de datos (TAD), que es usado mediante las operaciones definidas en el tipo
- Para los implementadores
  - Un fichero es una colección de bloques de disco que se presenta al usuario con una unidad lógica
  - Cada fichero tiene un identificador binario

Datos y atributos

- Los ficheros contienen datos y atributos
  - Los datos están formados por la secuencia de elementos de información que son escritos y leídos mediante las operaciones definidas para tal fin
  - Los atributos proporcionan información sobre el fichero (son metadatos)
- Atributos habituales de un fichero
  - Longitud, propietario, creador, información de protección, tipo de fichero, fecha de creación, fecha de último de acceso, fecha de última modificación, etc

Atributos de ficheros en UNIX

- En el sistema operativo UNIX los ficheros tienen los siguientes atributos
  - Propietario
  - Longitud
  - Tipo de fichero
    - Regular, directorio, enlace simbólico, fichero de dispositivo en modo bloque, fichero de dispositivo en modo carácter, socket
  - Fechas de último acceso y última modificación
  - Número de enlaces:
    - Número de nombres diferentes que tiene el fichero
  - Máscara de protección
    - Hay tres tipos de usuarios (propietario, grupo, resto de usuarios) y tres tipos de permisos (lectura, escritura, ejecución)

Atributos de ficheros en NT

- En Windows NT los datos son atributo más
- Algunos atributos de los ficheros son
  - Nombre del fichero
  - Tamaño del fichero
  - Sólo lectura
  - Fichero comprimido
  - El fichero es un directorio
  - Fichero temporal
  - Tipo de fichero (ordinario, tubería, dispositivo, ...)
  - Fechas de creación, última modificación y último acceso
  - Descriptor de seguridad

Fichero como tipo abstracto de datos

- Un fichero es un tipo abstracto de datos, sobre el que se definen una serie de operaciones
- Operaciones más habituales
  - Creación
  - Apertura
  - Escritura
  - Lectura
  - Cierre
  - Posicionamiento aleatorio del puntero de lectura/escritura
  - Eliminación del fichero
  - Renombrado

Operaciones básicas sobre ficheros en UNIX y Win32

Win32	UNIX	Descripción
CreateFile	open	Crea un fichero, o abre uno existente
DeleteFile	unlink	Elimina un fichero existente
CloseHandle	close	Cierra un fichero
ReadFile	read	Lee datos de un fichero
WriteFile	write	Escribe datos en un fichero
SetFilePointer	lseek	Posiciona el puntero del fichero en una posición específica
GetFileAttributes	stat	Devuelve los atributos de un fichero
LockFile	fcntl	Bloquea parte de un fichero
UnlockFile	fcntl	Desbloquea una parte de un fichero previamente bloqueada

Tipos de ficheros

- Cuando se utiliza el término “fichero” se suele hacer referencia a ficheros ordinarios o regulares
  - Son los que contienen información, en general
- Pero en un SO pueden existir otros tipos de ficheros
  - Son, en la mayoría de los casos, especializaciones de los ficheros ordinarios. Ejemplo: directorios
  - O son ficheros ordinarios cuyo contenido debe tener una estructura concreta. Ejemplos: ficheros ejecutables
- Los ficheros ordinarios también se suelen clasificar en dos tipos: ASCII o binarios

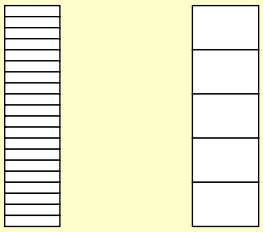
Tipos de ficheros

- Otra forma de distinguir tipos de ficheros es por su extensión (un sufijo del nombre del fichero)
  - Aunque para muchos sistemas operativos el nombre del fichero es independiente de su contenido

Nombre	Tipo
pila.c	Fichero que contiene un programa en C
pila.h	Fichero de cabecera en C
pila.o	Fichero que contiene un fichero compilado en C
Pila.java	Fichero que contiene un programa en Java
Pila.class	Fichero que contiene un programa compilado en Java
Pila.tar.gz	Fichero que contiene un fichero tar comprimido con gzip

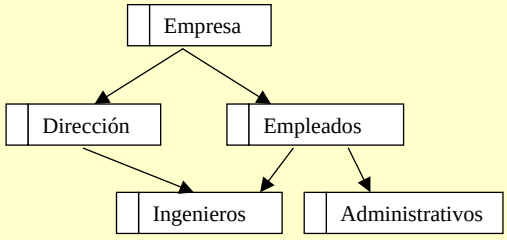
Estructura interna

- Los ficheros se pueden estructurar, desde un punto de vista lógico, de varias formas



Secuencia de bytes

Secuencia de registros



Árbol de registros

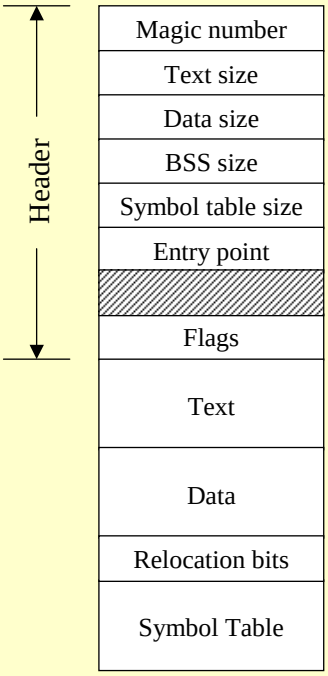
- La más usada hoy día es la secuencia de bytes
  - Es la que permite mayor flexibilidad

Estructura interna

- Independientemente de la estructura interna,
  - El contenido de un fichero también puede tener determinada estructura
- Ejemplo:
  - En UNIX y Windows NT los ficheros son secuencias de caracteres
  - Los directorios se implementan como ficheros
  - Como tales ficheros, son secuencias de bytes
  - Pero la información contenida en ellos debe reflejar un directorio, que es una tabla de estructuras

Estructura interna

- Ejemplo: ficheros ejecutables en UNIX



Métodos de acceso

- Existen varias formas de acceder la información que contiene un fichero
  - Acceso secuencial
  - Acceso directo
  - Acceso indexado

Directorios

- El número de ficheros que se puede almacenar en una unidad de disco es muy elevado
  - Es necesario poder organizarlos
- Los ficheros se organizan en dos niveles
  - En un primer nivel, el sistema de ficheros se divide en particiones (también denominadas volúmenes o incluso “sistemas de ficheros”, en UNIX)
  - En un segundo nivel, cada partición se organiza mediante directorios (también denominados carpetas)

Estructura lógica de un directorio

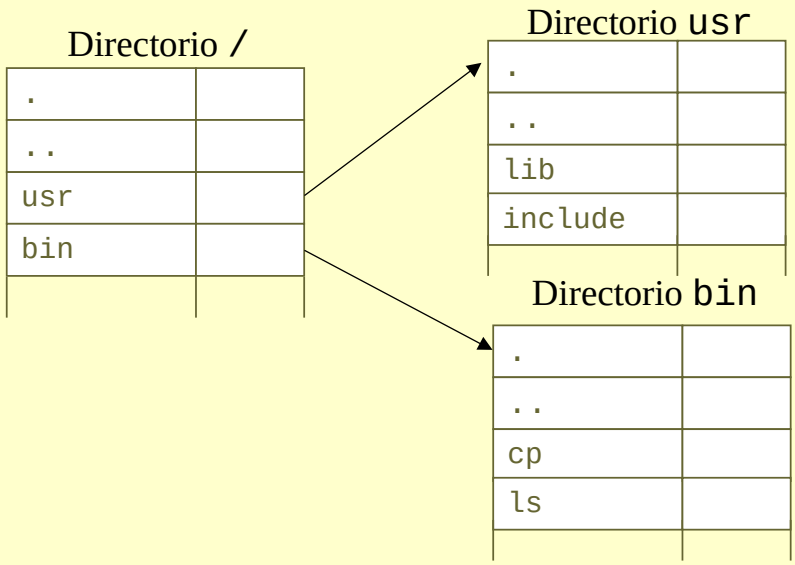
- Un directorio es una tabla en la que se asocia a un identificador ASCII la información necesaria para
  - Acceder a los atributos del fichero
  - Localizar los bloques de disco en los que el fichero está almacenado

Directorio

Nombre ASCII	Atributos y datos
.	
..	
pila.c	
pila.o	
pila	

Formas de organización

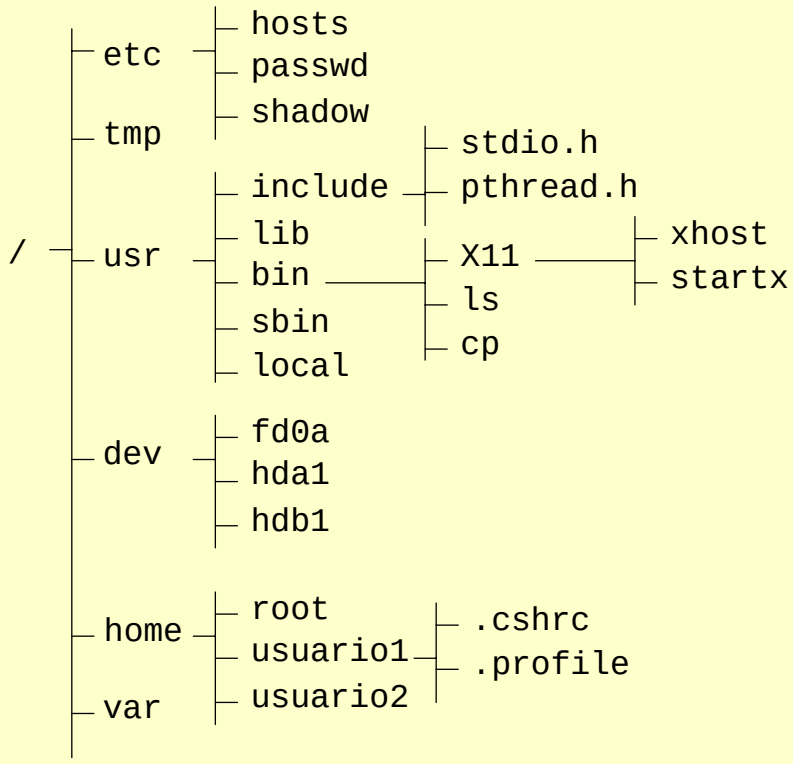
- Una entrada de directorio puede hacer referencia a otro directorio
  - Los directorios permiten, en general, organizar los ficheros en estructuras en forma de árbol



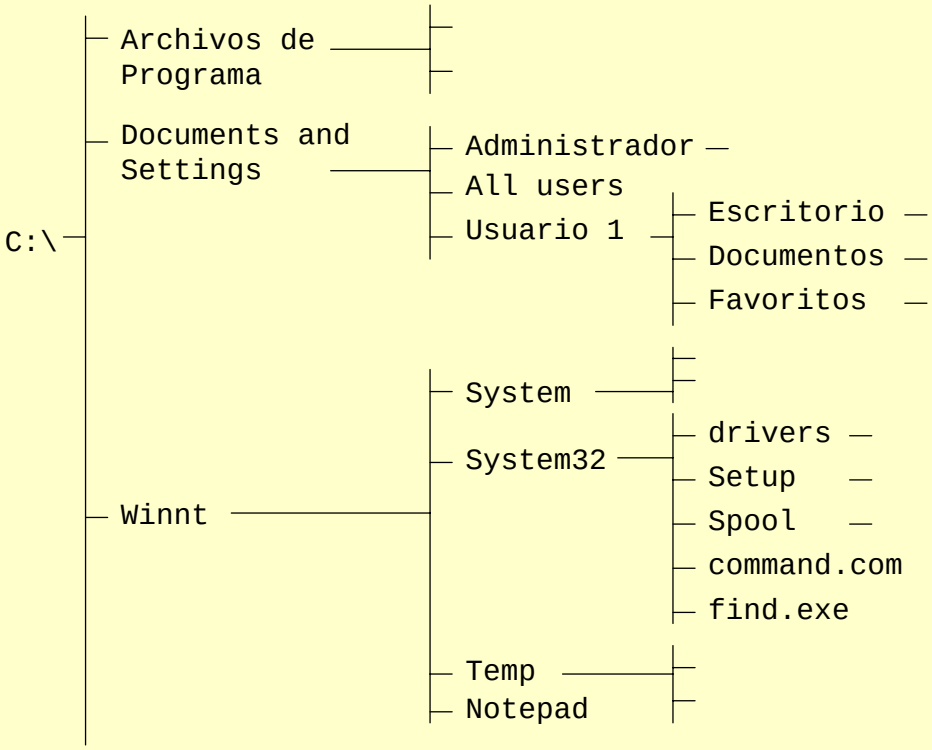
Formas de organización

- Las estructura en árbol es la más habitual
- En sistemas antiguos
  - Sólo existía un nivel de directorio. Ejemplo: CP/M
- En otros sistemas, la estructura en árbol se generaliza en un grafo
  - Ejemplo: UNIX

Organización en UNIX



Organización en Windows 2000



Ficheros compartidos

- Existen situaciones en las que es necesario compartir ficheros o directorios
  - El fichero compartido debería aparecer en más de un directorio a la vez
- Si el sistema operativo no permite esta posibilidad
  - La alternativa es copiar los ficheros a compartir
  - La modificación de una copia no afecta al resto
- En UNIX sí es posible asignar más de un nombre a un mismo fichero
  - Cada nombre se denomina enlace

Ficheros compartidos

- Dos tipos de enlaces en UNIX
  - *Hard link*: un enlace es un puntero a un fichero o directorio
  - *Symbolic link*: un enlace es un puntero a un fichero especial, cuyo contenido es la ruta del fichero a compartir
- Mediante enlaces la estructura de directorios en árbol se transforma en un grafo
  - El grafo es acíclico dirigido, si se comparten ficheros
  - Pero puede ser cíclico si se comparten directorios

Directorios como TAD

- Al igual que los ficheros, los directorios se pueden considerar como tipos abstractos de datos
- Operaciones más habituales
  - Crear directorio
  - Borrar un directorio vacío
  - Buscar el primer fichero en el directorio
  - Buscar el siguiente fichero
  - Mover un fichero de un directorio a otro
  - Cambiar el directorio de trabajo

Operaciones básicas sobre directorios en UNIX y Win32

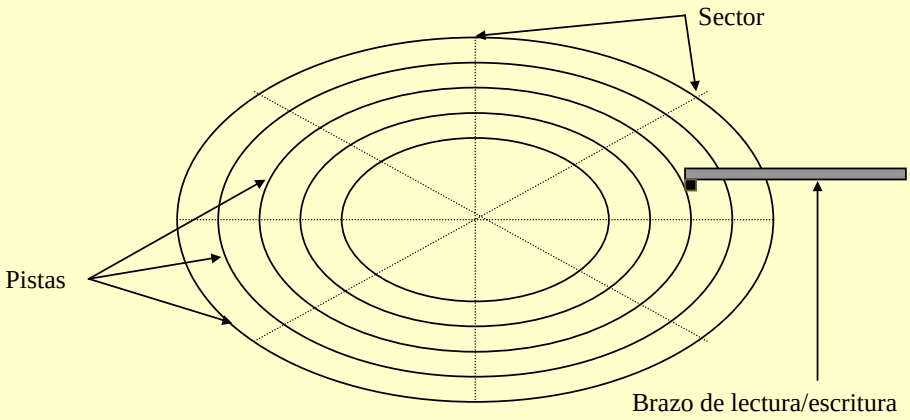
Win32	UNIX	Descripción
CreateDirectory	mkdir	Crea un directorio
RemoveDirectory	rmdir	Borra un directorio vacío
FindFirstFile	opendir	Prepara el directorio para leer la primera enterada
FindNextFile	readdir	Lectura de la siguiente entrada
MoveFile	rename	Mueve un fichero a otro directorio
SetCurrentDirectory	chdir	Cambia el directorio de trabajo

Implementación

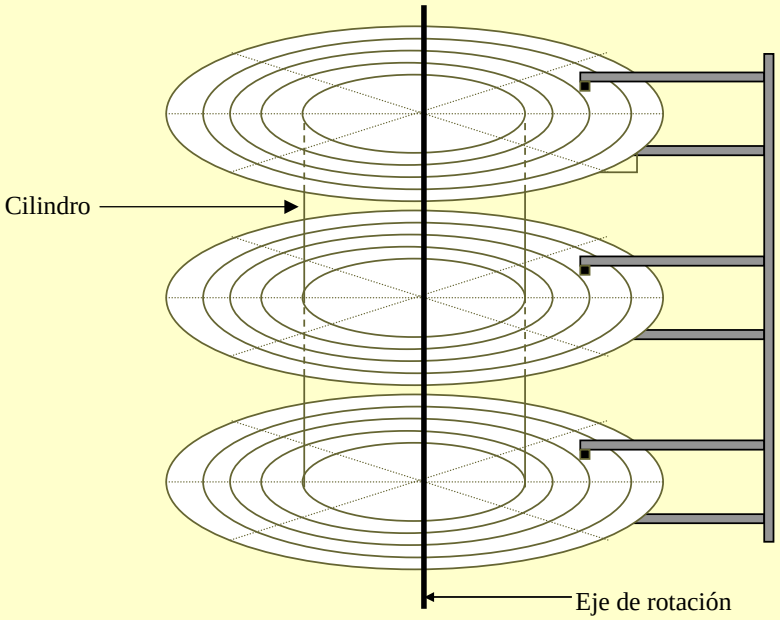
- Aspectos a considerar para implementar un sistema de ficheros
  - Almacenamiento de la información en los discos
  - Implementación de los ficheros
  - Implementación de los directorios
  - Gestión del espacio libre del disco
  - Fiabilidad
  - Rendimiento

Arquitectura de un disco

- Los discos se estructuran en torno a pistas y sectores
  - La unidad mínima de lectura/escritura es el sector
  - Para aumentar el rendimiento, en lugar de sectores individuales se suelen usar grupos de sectores contiguos, denominados bloques o clusters



Arquitectura de un disco con múltiples platos



Tiempos de acceso

- El tiempo requerido para acceder a un bloque de disco depende de tres factores
  - Tiempo de búsqueda
    - Tl tiempo que tarda la cabeza lectora/escritora en posicionarse en la pista adecuada
  - Tiempo de latencia
    - Tiempo que tarda el inicio del sector a ser accedido en pasar por debajo de la cabeza lectora/escritora
  - Tiempo de transferencia
    - Tiempo en transferir la información del disco a memoria
- El tiempo mayor es el de búsqueda
  - La reducción de este tiempo es fundamental para obtener un buen rendimiento

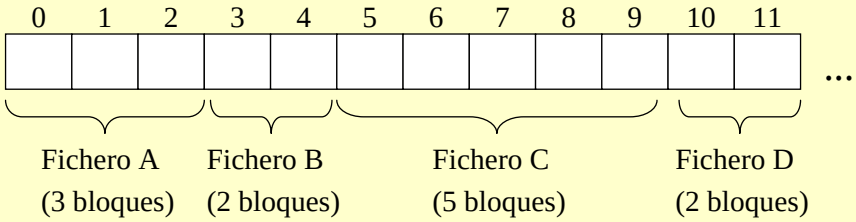
Implementación de ficheros

- Aspecto más importante a considerar
  - Cómo se determina qué bloques del disco pertenecen a un fichero
  - Depende de cómo se asignen los bloques del disco a los ficheros
- Hay cuatro métodos de asignación básicos
  - Asignación contigua
  - Asignación mediante lista encadenada
  - Asignación mediante lista encadenada almacenada en una tabla
  - Nodos índice

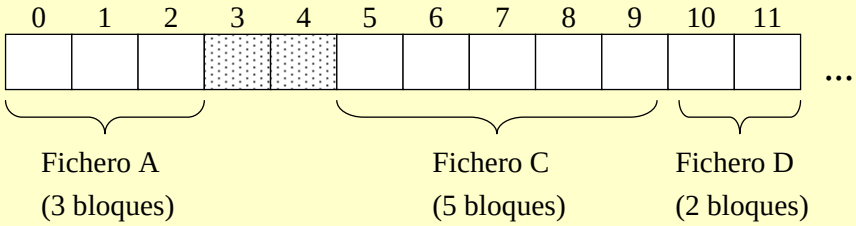
Asignación contigua

- Fundamento
  - Cada fichero se almacena en un conjunto de bloques contiguos

Estado Inicial



Estado tras borrar el fichero B



Francisco López Valverde
Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga

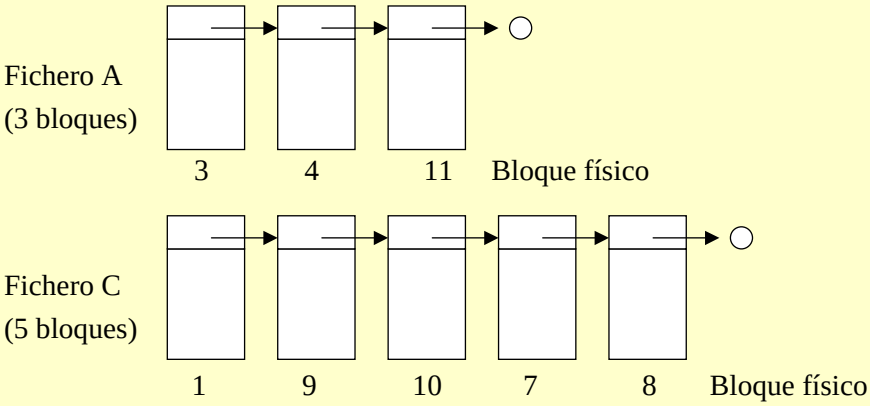
Asignación contigua

- Ventajas
  - Simplicidad: para saber qué bloques pertenecen a un fichero basta conocer el bloque inicial y el número de bloques del fichero
  - Rendimiento: se puede leer un fichero entero en una simple operación
- Inconvenientes
  - Fragmentación externa
  - Imposibilidad de conocer a priori el tamaño de los ficheros
- Aplicaciones: CD/ROM, DVD

Francisco López Valverde
Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga

Asignación mediante lista encadenada

- Fundamento
  - Cada fichero se almacena en una lista encadenada de bloques. La primera palabra de un bloque es un puntero al siguiente bloque

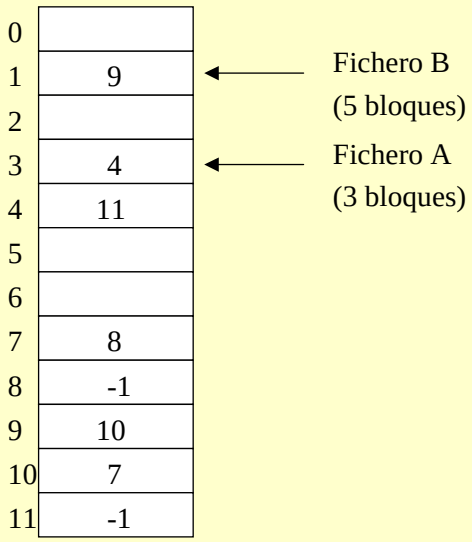


Asignación mediante lista encadenada

- Ventajas
  - Simplicidad: para saber qué bloques pertenecen a un fichero basta conocer el bloque inicial y el número de bloques del fichero
  - No existe fragmentación interna
- Inconvenientes
  - Acceso lento
  - Problemas si se estropea un bloque
  - El contenido de un bloque no es potencia de dos

Asignación mediante lista encadenada en una tabla

- Fundamento
  - Los punteros se almacenan en un tabla (*File Allocation Table* o FAT)



Francisco López Valverde
Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga

SISTEMAS OPERATIVOS
Tema 1: Sistemas de Ficheros

Introducción
Ficheros
Directorios
Implementación
- Almacenamiento en discos
- Implementación de ficheros
- Implementación de directorios
- Gestión del espacio en disco
- Fiabilidad
- Rendimiento

Sistemas de Ficheros en UNIX
Sistemas de Ficheros en Windows 2000/XP
Aspectos avanzados
Fuentes de información

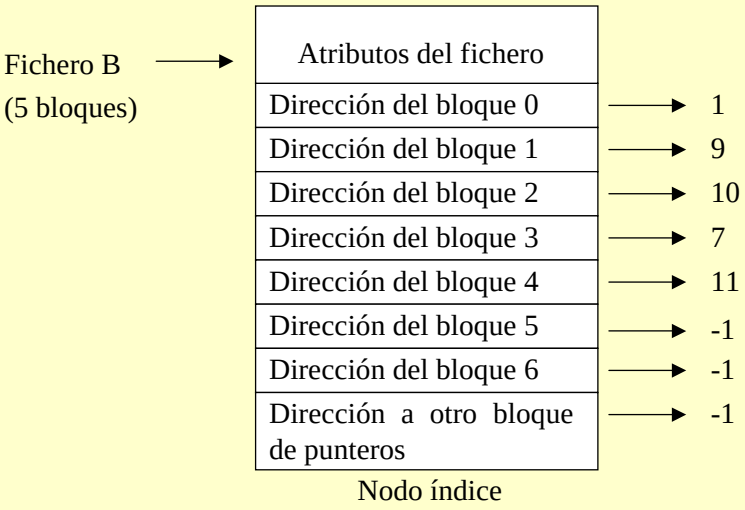
Asignación mediante lista encadenada en una tabla

- Ventajas
  - Simplicidad: para saber qué bloques pertenecen a un fichero basta conocer el bloque inicial y el número de bloques del fichero
  - Mayor rendimiento porque los punteros están en memoria
- Inconvenientes
  - Toda la tabla debe estar en memoria
  - Su tamaño es proporcional al tamaño de los discos
- Ejemplos
  - MS-DOS
  - Windows 98/Me

Francisco López Valverde
Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga

Asignación mediante nodo índice

- Fundamento
  - Los punteros se almacenan en una estructura denominada nodo índice, que contiene los atributos y las direcciones de los bloques de datos del fichero



Nodo índice

Asignación mediante nodo índice

- Ventajas
  - Los nodos índice sólo han de estar en memoria cuando los ficheros a los que representan son abiertos
  - Su tamaño depende del número máximo de ficheros abiertos, no del tamaño del disco
- Inconvenientes
  - Algunas operaciones son costosas de realizar
    - Borrar partes de un fichero
    - Añadir o borrar datos en el fichero en cualquier lugar que no sea el final del mismo
- Ejemplo:
  - UNIX

Implementación de directorios

- Desde un punto de vista lógico
  - Un directorio es una tabla que permite poder acceder a los atributos y los bloques de datos de un fichero a partir de su nombre en formato de cadena de caracteres

Directorio

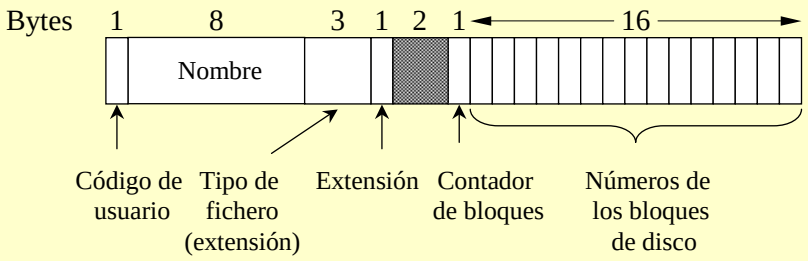
Nombre ASCII	Atributos y datos
.	
..	
pila.c	
pila.o	
pila	

Implementación de directorios

- Esta estructura lógica se puede implementar de varias formas
  - Incluyendo los atributos y las direcciones de los bloques de datos en la misma entrada de directorio
  - Almacenando los atributos en la misma entrada de directorio y las direcciones de los bloques de datos en una estructura externa
  - Almacenando tanto los atributos como las direcciones de los bloques de datos en una estructura externa

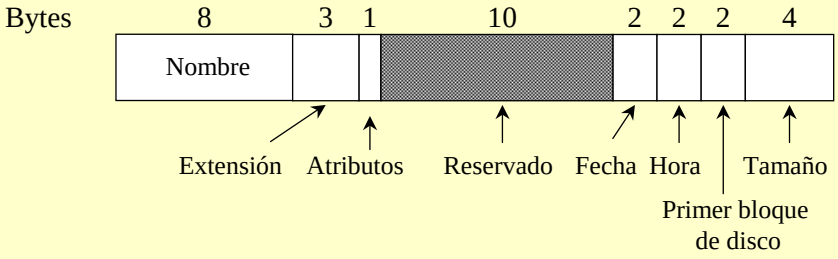
Implementación de directorios

- Los atributos y las direcciones de los bloques de datos se almacenan en la misma entrada de directorio
  - Ejemplo: CP/M



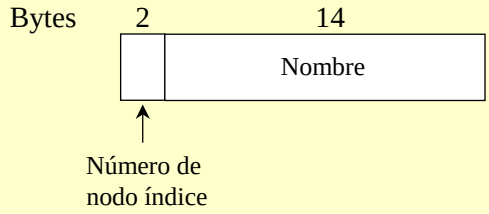
Implementación de directorios

- Los atributos se almacenan en la misma entrada de directorio y las direcciones de los bloques de datos en una estructura externa
  - Ejemplo: MS-DOS



Implementación de directorios

- Tanto los atributos como las direcciones de los bloques de datos se almacena en una estructura externa
  - Ejemplo: UNIX (sistemas anteriores a BSD)



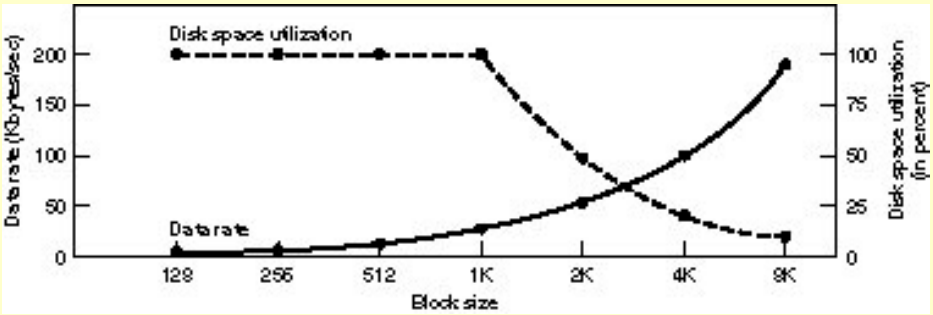
Gestión del espacio en disco

- Para almacenar ficheros en un disco
  - Su contenido se reparte habitualmente en bloques (la excepción es el método de asignación contigua)
- Factores a considerar
  - El tamaño que deben tener los bloques
  - Cómo gestionar los bloques que quedan libres

Tamaño de los bloques

- Intuitivamente
  - Un tamaño de bloque excesivamente pequeño permite aprovechar el espacio del disco, pero penaliza el rendimiento porque hay que hacer muchos accesos a disco para leer o escribir un fichero
  - Un tamaño excesivamente grande aumenta el rendimiento, pero se desperdicia espacio debido a la fragmentación externa
- Estudios realizados sobre sistemas UNIX y NT
  - Muestran que el tamaño medio de un fichero está hoy día en torno a 2 KB
  - El tamaño de bloque debería estar en torno a 2KB

Tamaño de los bloques



Gestión de los bloques libres

- Para determinar qué bloques están libres hay dos técnicas básicas
  - Lista encadenada
  - Mapa de bits
- Lista encadena de bloques libres
  - Es una lista de bloques de disco que contienen números de bloques libres
  - Se suelen usar bloques libres para almacenar esta lista
- Mapa de bits
  - Cada bit representa un bloque
  - Si está a 1, el bloque está libre, y ocupado en caso contrario

Gestión de los bloques libres

- Espacio requerido
  - Se asume tamaño de bloque de 1KB, direcciones de bloques de 32 bits y un disco de 16GB
  - Lista encadenada: hasta 16.794 bloques ???????
  - Mapa de bits: 2048 bloques
- Espacio requerido en memoria
  - En el caso de lista encadena, sólo debe estar el primer bloque de la lista
  - En el caso del mapa de bits, éste debe estar entero en memoria
- Ventaja de usar un mapa de bits
  - Se facilita la búsqueda de bloques libres adyacentes

Fiabilidad

- La información contenida en un disco se puede perder por diversas causas
  - Borrado accidental
  - Rotura del disco
- Técnicas que se pueden utilizar
  - Realización de copias de seguridad (*backups*)
  - Verificación de la consistencia del sistema de ficheros
  - Utilización de técnicas RAID

Copias de seguridad

- Idea básica
  - Almacenar en un dispositivo externo una copia de los ficheros que son críticos
- Dispositivos utilizables
  - CD-R (han reemplazado a los disquetes)
  - Cintas
  - Discos extraíbles
  - Discos remotos accesibles a través de una red

Copias de seguridad

- Aspectos a considerar
  - La realización de las copias de seguridad lleva mucho tiempo. Se han de copiar sólo los ficheros necesarios
  - No es necesario hacer una copia de los ficheros que no han sido modificados desde la última. Hay que hacer copias incrementales (*incremental dumps*)
  - Comprimir los ficheros al hacer la copia introduce un factor de riesgo
  - Es difícil hacer copias sin detener el sistema
  - Problemas de seguridad

Copias de seguridad

- Herramientas disponibles en Unix:
  - `cpio`
    - Almacena estructuras de directorio en un fichero
    - Válido para salvar pequeñas cantidades de datos
  - `tar`
    - Almacena estructuras de directorio en una cinta
    - Válido para salvar pequeñas cantidades de datos
  - `dump`
    - Diseñado para hacer backups totales e incrementales
    - Lo usan los administradores de sistemas

Consistencia del sistema de ficheros

- Si un bloque es modificado en memoria pero no se ha escrito en disco se produce una inconsistencia
  - El problema se agrava si los bloques forman parte de estructuras críticas, como directorios, la lista de bloques libres o nodos índice
- La mayoría de los sistemas operativos ofrecen utilidades para comprobar la consistencia del sistema de ficheros
  - `fsck`, en Unix
  - `scandisk`, en Windows
- Se puede comprobar la consistencia
  - A nivel de bloques
  - A nivel de ficheros

Consistencia del sistema de ficheros

- Ejemplo: comprobación de la consistencia de bloques en Unix
- Son necesarias dos tablas
  - Ambas contienen un contador por bloque
  - Los contadores están inicializados a cero
- Misión de los contadores
  - Los de la primera tabla cuentan las veces que un bloque está presente en un fichero
  - Los de la segunda cuentan las veces que un bloque está presenta en la lista de bloques libres
- Funcionamiento
  - El programa examina los nodos índice y la lista de bloques libres e incrementa los contadores correspondientes

Comprobación de la consistencia en Unix

- Pueden darse cuatro situaciones
  - Cada bloque tiene un uno en una de las dos tablas
    - En este caso el sistema es consistente
  - Un bloque tiene un cero en ambas tablas
    - Es un bloque perdido (missing block)
    - El problema se soluciona añadiéndolo a la lista de bloques libres
  - Un bloque aparece más de una vez en la lista de bloques libres
    - Se soluciona reconstruyendo la lista
  - Un bloque aparece más de una vez en la lista de bloques usados
    - Error grave que es notificado al administrador del sistema

Técnicas RAID

- RAID (*Redundant Array of Inexpensive Disks*) es un dispositivo compuesto típicamente por
  - Un array de discos
  - Un controlador (habitualmente SCSI)
- Se pueden usar para aumentar
  - La fiabilidad
  - El rendimiento
- Existen siete esquemas RAID
  - Se denominan niveles, aunque no son jerárquicos
  - El nivel más bajo es el cero y el más alto el seis

Técnicas RAID

- RAID nivel 0
  - Todos los discos aparecen como uno sólo
  - No tiene redundancia
  - Permite acceder en paralelo a varios bloques de un mismo fichero
- RAID nivel 1
  - La información se duplica en dos discos
  - Si un disco falla, existe una copia en otro
  - Se suele denominar *mirroring*
  - Las escrituras son algo más lentas
  - El rendimiento de las lecturas puede aumentar hasta dos veces
  - Se utiliza habitualmente en pequeños servidores

Técnicas RAID

- RAID nivel 2
  - Trabaja a nivel de bytes, añadiendo bits adicionales para usar un código Hamming
  - Los bits de cada byte están repartidos en discos distintos, incluidos los adicionales
  - Implica acceso sincronizado a los discos y el cómputo de la función Hamming en cada acceso
  - Permite el acceso en paralelo, pero sólo se puede realizar una operación de E/S a la vez. Aporta ventajas al acceder a grandes ficheros
- RAID nivel 3
  - Es una simplificación del nivel 2
  - Usa un disco que contiene bits de paridad

Técnicas RAID

- RAID nivel 4
  - Es como el nivel 0, pero añade un disco redundante con información de paridad a nivel de bloque
  - Permite realizar operaciones en paralelo, pero cada escritura implica un acceso al disco de bits de paridad, que puede constituir un cuello de botella
- RAID nivel 5
  - Es como el nivel 4, pero la información de paridad está distribuida entre los distintos discos
  - De esta forma se elimina el problema de cuello botella del disco de paridad

Técnicas RAID

- RAID nivel 6
  - Es similar al nivel 5, pero realizan dos cálculos de paridad en lugar de uno. Cada resultado se almacena en un disco distinto
  - Es necesario un disco más que en el nivel 5
  - Las escrituras son muy lentas, pero proporciona un grado de fiabilidad muy elevado

Rendimiento

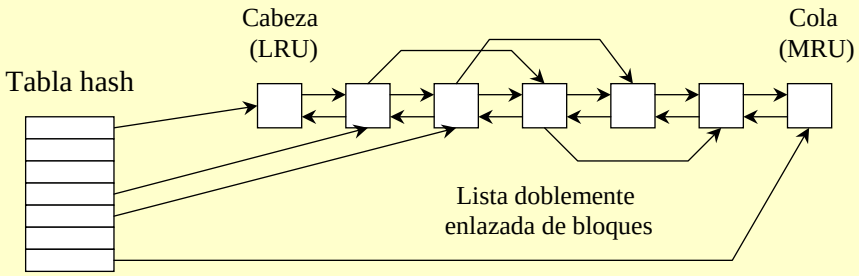
- En la actualidad
  - El tiempo de acceso de la memoria RAM es de pocos nanosegundos; el tiempo de acceso a un disco es del orden de varios milisegundos
- Conclusión
  - El disco es aproximadamente un millón de veces más lento que la memoria, por lo que necesario la aplicación de técnicas que permitan aumentar el rendimiento de los discos
- Técnicas aplicables
  - Técnicas de caché
  - Lectura anticipada
  - Reducción de los desplazamientos de la cabeza lectora/escritora

Técnicas de caché

- En el contexto de sistemas de ficheros
  - Una caché es una colección de bloques que lógicamente pertenecen a un disco, pero que se hayan localizados en memoria para aumentar el rendimiento
- Funcionamiento básico
  - Siempre que se necesita un bloque se busca primero en la caché
  - Si está, se ha ahorrado un acceso a disco
  - Si no está, se busca en el disco, se carga en la caché, y es accedido en memoria

Técnicas de caché

- Aspectos de implementación
  - Debido a que suelen existir muchos bloques en la caché, se usan tablas hash para encontrarlos rápidamente
  - Si la caché se llena, hay que aplicar alguna técnica de reemplazo, tipo LRU



Técnicas de caché

- La aplicación de una política LRU pura no es deseable
  - Los bloques que contienen nodos índice o directorios son esenciales para la consistencia del sistema de ficheros, por lo que deberían ser escritos a disco inmediatamente cuando son modificados
  - Hay bloques cuya probabilidad de ser usados prontamente es elevada, como los bloques parcialmente llenos de ficheros abiertos en modo escritura, por lo que deberían estar al final de la lista

Técnicas de caché

- Para evitar inconsistencias
  - No es deseable que un bloque modificado esté en memoria muchas veces
- caché *write-through*
  - Cuando un bloque se modifica, es escrito en disco
  - Empleado en MS-DOS
- caché con escritura retardada
  - Se espera un tiempo antes de escribir en disco los bloques modificados
  - En UNIX existe una llamada denominada sync que realiza esa función, y es invocada automáticamente cada 30 segundos

Lectura anticipada

- Fundamentos
  - Intentar colocar en la caché los bloques antes de éstos sean solicitados
- Ejemplo: lectura secuencial de un fichero
  - Cuando un proceso solicita un bloque, el sistema comprueba si el siguiente bloque está en la caché
  - Si no está, lo busca
- La técnica no es válida para accesos aleatorios a ficheros

Reducción de los desplazamientos

- Reducir los movimientos de la cabeza lectora/escritora es fundamental para el rendimiento del sistema ficheros
- Idea básica
  - Situar los bloques que probablemente sean accedidos en secuencia cerca unos de otros, a ser posible dentro del mismo cilindro
- Técnicas que se pueden aplicar
  - *Clustering*
  - Situar los nodos índice (o estructuras equivalentes) en el centro del disco, en lugar del principio
  - Dividir el disco en grupos de cilindros

Sistemas de Ficheros en UNIX

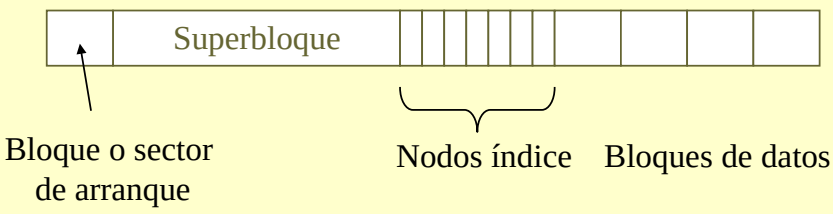
- Aspectos de implementación
- Llamadas al sistema: ficheros
- Llamadas al sistema: directorios

Sistemas de Ficheros en UNIX

- Definición genérica de sistema de ficheros
  - Parte del sistema operativo responsable de la gestión de los ficheros y directorios
- En UNIX, en cambio
  - Un sistema de ficheros es una unidad de almacenamiento que contiene una estructura de directorios y ficheros independiente
  - Puede ser físico (disco) o lógico (partición)

Estructura del Sistema de Ficheros en UNIX

- Estructura de un sistema de ficheros clásico en UNIX



Estructura del Sistema de Ficheros en UNIX

- Sector de arranque
  - Se usa para iniciar la ejecución del sistema operativo
- Superbloque
  - Bloque que contiene la información relevante del sistema de ficheros
- Nodos índice
  - Estructuras que contiene información sobre los ficheros
- Bloque de datos
  - Son los bloques de disco que contienen la información de los ficheros

Estructura del Sistema de Ficheros en UNIX

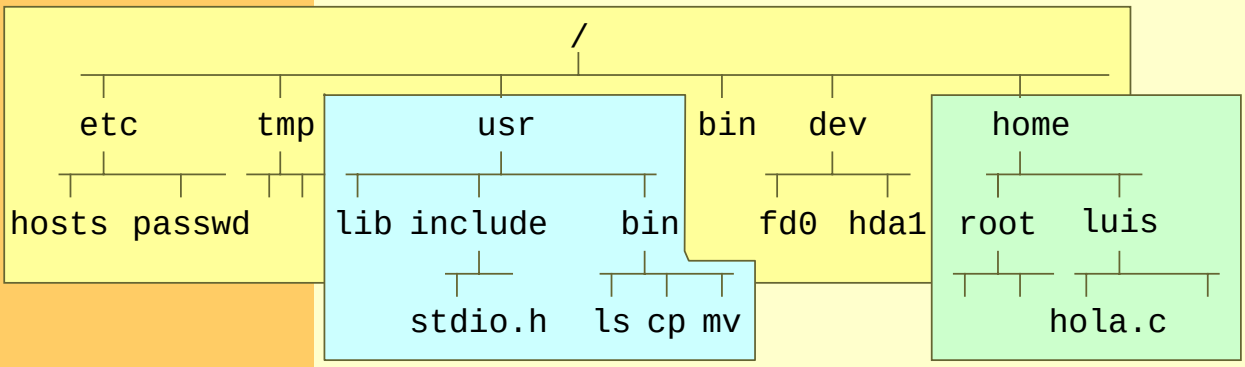
- Todo fichero está representado por un nodo índice
  - El número total de nodos índice indica el número máximo de ficheros que puede tener un sistema UNIX
- Cualquier modificación sobre un fichero
  - Implica una actualización de los datos del nodo índice
- El superbloque registra información acerca de los nodos índice y los bloques de datos
  - Esta información se modifica al realizar operaciones sobre los ficheros

Rendimiento

- Para aumentar el rendimiento
  - Existe una copia del superbloque y los nodos índice en memoria
  - Los cambios en memoria no se propagan inmediatamente al disco
  - Esto implica que NO se debe apagar un sistema UNIX directamente, pulsando el interruptor de encendido

Estructura del Sistema de Ficheros en UNIX

- A diferencia de otros sistemas operativos
  - Aunque existan varios sistemas de ficheros, el usuario únicamente percibe una única estructura de directorio



Dispositivos

- En Unix los dispositivos se tratan como ficheros
  - Los ficheros para manejar dispositivos se almacenan en el directorio `/dev`
- Existen dos tipos de dispositivos
  - Dispositivos de bloques: se leen y escriben bloques de datos (discos, cintas)
  - Dispositivos de caracteres: se leen y escriben caracteres individuales (teclado, ratón)

Dispositivos. Rendimiento

- Para acelerar el rendimiento de los dispositivos en modo bloque
  - Existe una cache (*buffer cache*) en el sistema de ficheros
- Las modificaciones sobre los ficheros
  - No se transmiten inmediatamente a disco, sino que los bloques modificados permanecen un tiempo en la cache
  - Periódicamente, un demonio graba los bloques modificados a disco
  - Si durante ese tiempo se apaga el sistema se producen inconsistencias

Dispositivos. Rendimiento

- Si se invoca la instrucción sync
  - Se escriben a disco todas las estructuras que están en memoria y han sido modificadas (bloques de datos, nodos índice, superbloque)
- Como regla general para evitar inconsistencias
  - Nunca apagar un sistema UNIX pulsando el interruptor de encendido

Integración de dispositivos

- La integración un nuevo dispositivo de almacenamiento (disco, CD-ROM, floppy, etc) requiere:
  - Si el dispositivo no tiene un sistema de ficheros, hay que crear uno (formatear)
  - Integrar el nuevo sistema de ficheros mediante una operación denominada *montaje*

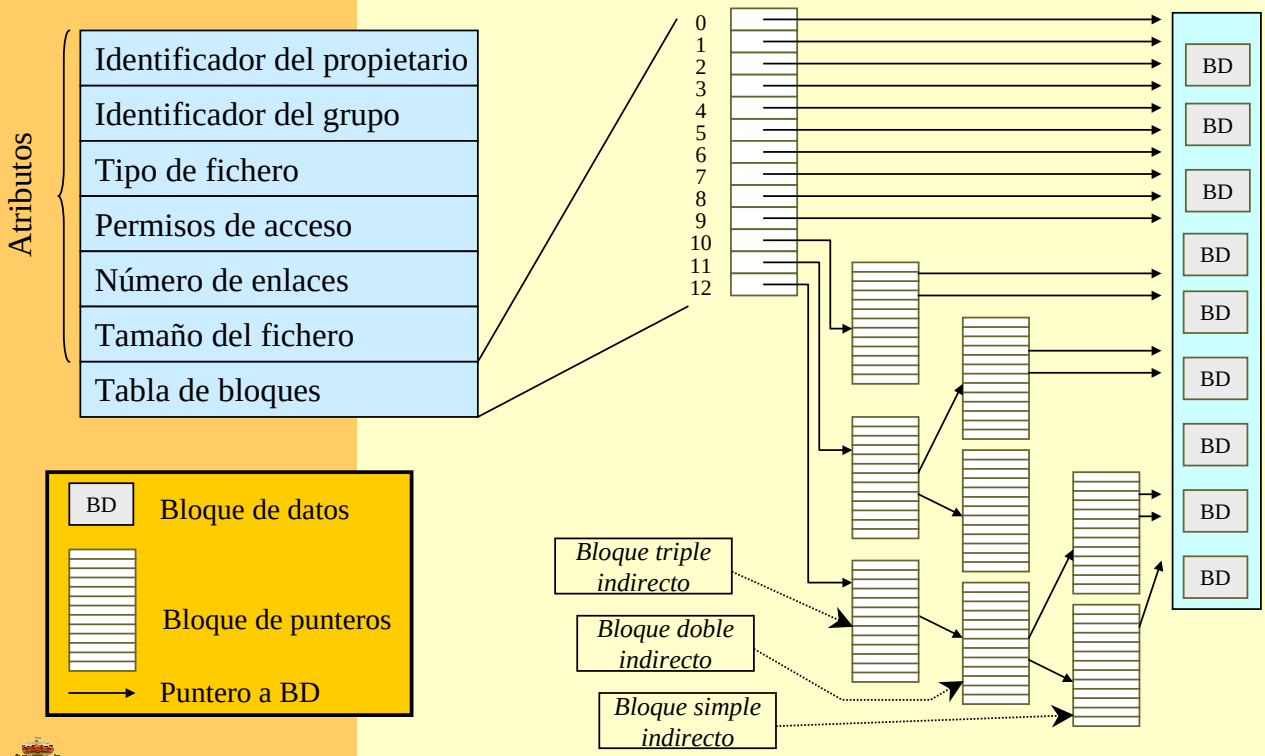
Integración de dispositivos

- Para montar un nuevo sistema de dispositivo
  - Se usa la orden `mount`
- Para desmontarlo
  - La instrucción es `umount`
- Ejemplo (en Linux):

```
# mount -t vfat /dev/hda1 /mnt/windows
```

- La instrucción `mount`, sin argumentos, indica las unidades que están montadas
- La instrucción `df` muestra información sobre los sistemas de ficheros

Estructura de un Nodo Índice



Nodos Índice. Ventajas

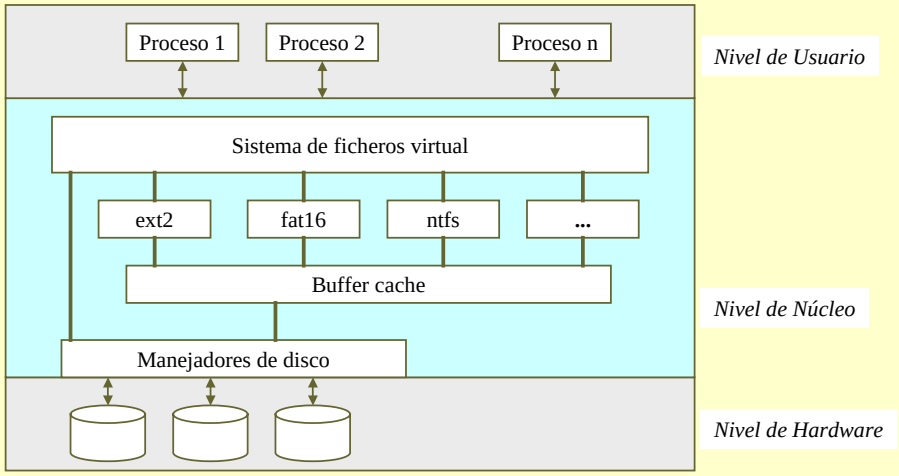
- Ventajas de usar nodos índice
  - Toda la información sobre el fichero está localizada en una única estructura, que se carga a memoria cuando va a ser usada
  - La mayoría de los ficheros en un sistema Unix son de pequeño tamaño
    - La información de los punteros directo es suficiente para conocer dónde está el fichero en disco
  - Los punteros a bloques indirectos permiten tener ficheros muy grandes

Nodos Índice. Inconvenientes

- Inconvenientes de usar nodos índice
  - Hay operaciones sobre ficheros que son costosas de realizar, por lo que no se permiten:
    - Insertar bytes en cualquier posición que no sea el final del fichero
    - Borrar bytes de un fichero
    - Truncar el tamaño de un fichero a un valor distinto de cero bytes
  - Operaciones que sí se permiten son
    - Leer o escribir cualquier byte del fichero
    - Añadir bytes al final del fichero, con el consiguiente aumento de tamaño
    - Truncar el tamaño del fichero a cero bytes

El Sistema de Ficheros de Linux

- Se denomina ext2
  - Aunque Linux admite otros sistemas de ficheros



Implementación de directorios

- Los directorios son ficheros
  - Pueden ser abiertos en modo lectura
  - Se pueden usar las llamadas `read`, `lseek`, `fstat` y `close`
- Pero son ficheros especiales
  - No pueden ser creados con `open` ni abierto en modo escritura
- Motivo:
  - La implementación de directorios puede variar de un sistema a otro

Implementación de directorios

- Los programas que hacen uso de directorios hacen uso de una estructura básica:
  - `struct dirent`
- Esta estructura contiene, entre otros, los siguientes campos:

Campo	Descripción
`d_ino`	Número de nodo índice
`d_name[]`	Nombre del fichero

Tablas para el uso de ficheros

- Para trabajar con ficheros el sistema operativo UNIX hace uso de tres tablas
  - La tabla de nodos índice
  - La tabla de ficheros
  - La tabla de descriptores de ficheros
- La tablas de nodos índice y de ficheros
  - Son tablas globales
- La tabla de descriptores de ficheros
  - Es local a cada proceso

Tablas para el uso de ficheros

- Las relaciones entre las tres tablas es la siguiente:

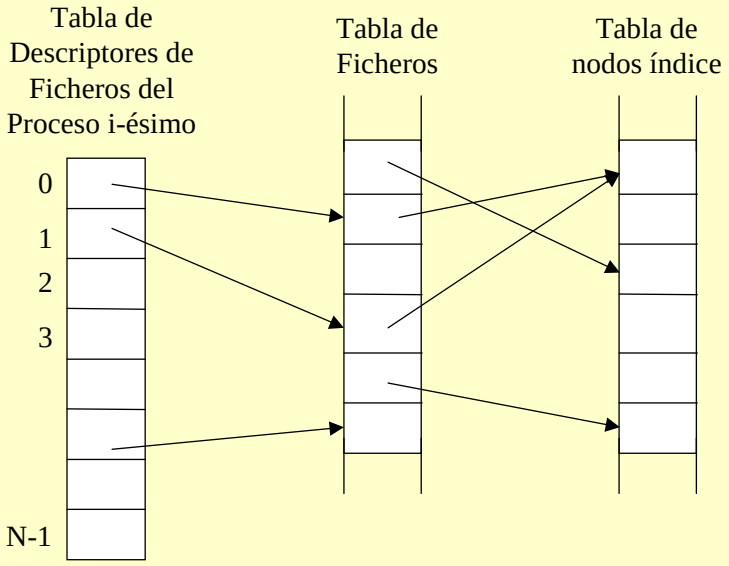


Tabla de descriptores de fichero

- La tabla de descriptores de fichero
  - Contiene una entrada por cada fichero abierto por el proceso
  - Los procesos acceden a los nodos índice de forma indirectamente, a través de los descriptores de ficheros
  - Tradicionalmente tenía un tamaño máximo de 20 entradas. En la actualidad depende del sistema, pero su límite puede ser de varias miles de entradas

Tabla de descriptores de fichero

- Las tres primeras entradas son especiales
  - El descriptor 0 hace referencia a la entrada estándar
  - El descriptor 1 hace referencia a la salida estándar
  - El descriptor 2 hace referencia a la salida estándar de error
- Estos tres descriptores
  - Son abiertos de forma automáticamente por el sistema cuando un proceso inicia su ejecución

Tabla de descriptores de fichero

- Asignación de descriptores
  - Los descriptores son devueltos por las llamadas al sistema que crean y abren ficheros
  - También son usados para acceder a dispositivos y a canales de comunicaciones
- Regla de asignación
  - Cuando se solicita un descriptor, se devuelve siempre el primero que está libre, comenzando desde 0
- Liberación de descriptores
  - Dado que la tabla tiene tamaño fijo, en conveniente devolver los descriptores cuando éstos no sean necesarios
  - La llamada habitual es `close()`

Tabla de ficheros

- La tabla de ficheros es una tabla global
  - Cada entrada contiene información sobre un fichero abierto
- Cada entrada contiene, entre otra información
  - Puntero al nodo índice asociado
  - Desplazamiento (*offset*) del puntero de lectura/escritura
  - Permisos de acceso del proceso que usa el fichero
  - Indicadores del modo de apertura del fichero
  - Contador de referencias a la entrada desde tablas de descriptores de ficheros

Tabla de nodos índice

- La tabla de nodos índice es una tabla global
  - Cada entrada contiene un nodo índice de un fichero
- Cada entrada contiene, entre otra información
  - Los atributos del fichero
  - Un contador de referencias a la entrada desde entradas de la tabla de ficheros
  - Estado del nodo índice en memoria (si ha sido modificado, si está bloqueado, etc)

Ejemplo de uso de las tres tablas

- El esquema de tres tablas usado
  - Permite un manejo flexible de los ficheros
- Ejemplo:
  - Paso 1: Un proceso A abre los siguientes ficheros

```
fd1 = open("/etc/passwd", O_RDONLY) ;
fd2 = open(".cshrc", O_RDWR) ;
fd3 = open("/etc/passwd", O_WRONLY) ;
```

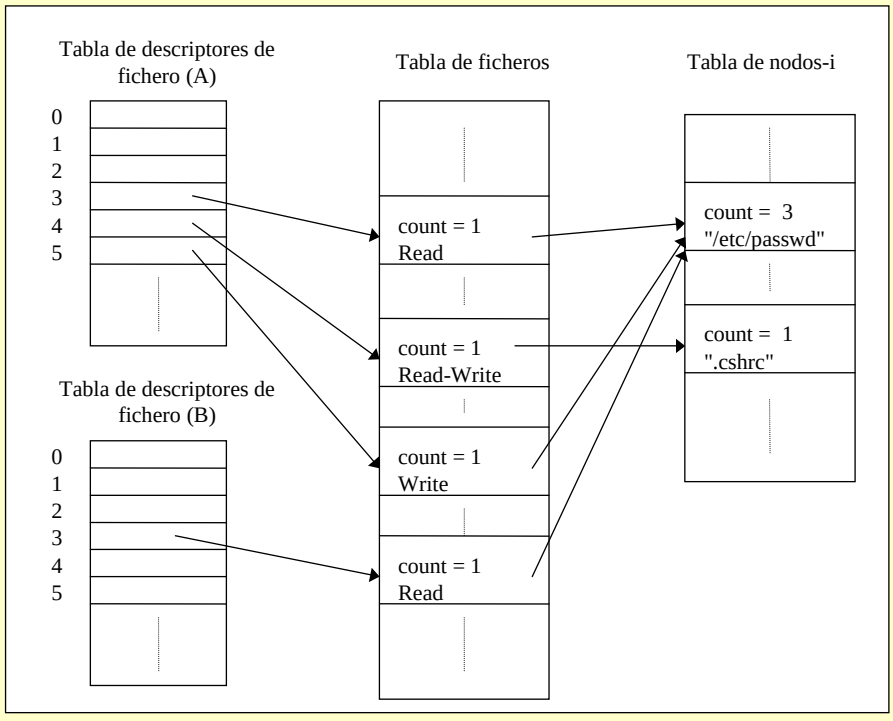
  - Paso 2: Un proceso B abre a continuación el siguiente fichero

```
fd1 = open("/etc/passwd", O_RDONLY) ;
```

  - Paso 3: El proceso A crea un proceso hijo, C

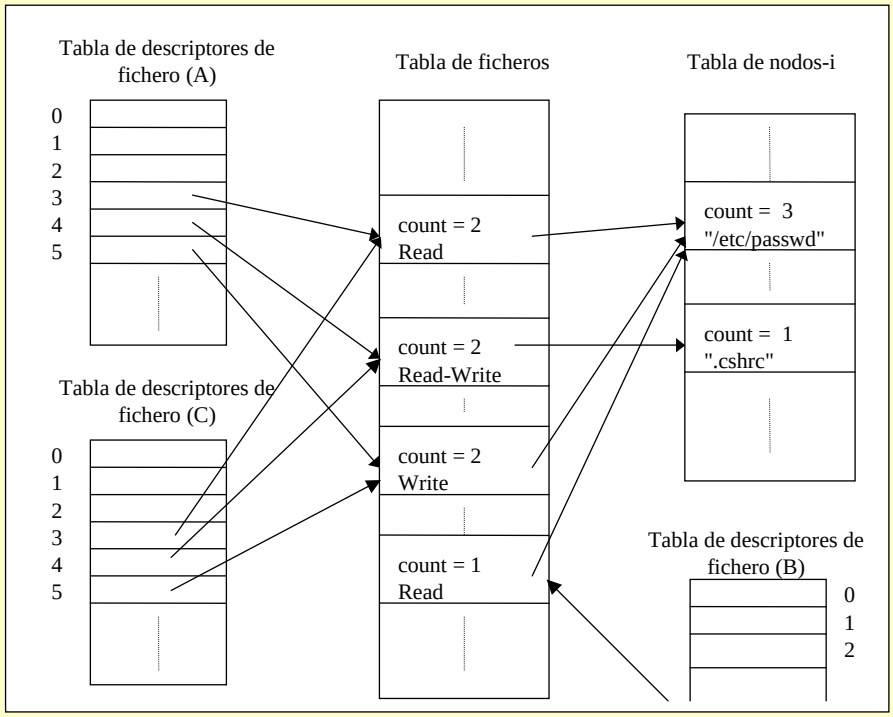
Ejemplo de uso de las tres tablas

- Situación tras los pasos 1 y 2



Ejemplo de uso de las tres tablas

- Situación tras el paso 3



Propietarios

- Todo fichero en UNIX
  - Lleva asociados un propietario (*uid*) y un grupo (*gid*)
- Todo proceso en UNIX lleva asociados dos propietarios
  - Propietario efectivo (*euid*), que es el que usa para determinar los privilegios
  - Propietario real
  - Habitualmente coinciden los dos tipos
- El concepto de propietario efectivo
  - Permite que cuando un usuario ejecute el fichero `/bin/passwd` pueda modificar el fichero `/etc/passwd`, que pertenece al superusuario

Permisos y modos

- Todo fichero
  - Tiene asociada una máscara de modo, que indica los permisos (lectura, escritura y ejecución) puede tener cada tipo de usuario (propietario, grupo y resto de usuarios)
- Además, para ficheros ejecutables existen permisos adicionales

Indicador	Descripción
S_ISUID	Cambia el usuario efectivo del proceso al usuario propietario del fichero
S_ISGID	Cambia el grupo efectivo del proceso al grupo propietario del fichero
S_ISVTX	Conocido como *sticky bit*, permite compartir el código del fichero por varios procesos

Permisos y modos

- La máscara de modo
  - También contienen bits que indican el tipo del fichero
  - Para tener acceso a esta información se proporcionan un conjunto de macros

Macro	Descripción
S_ISREG	Fichero regular
S_ISDIR	Directorio
S_ISCHR	Dispositivo de caracteres
S_ISBLK	Dispositivo de bloques

Llamadas al sistema: ficheros

- Operaciones básicas que se pueden realizar con ficheros

Llamada	Descripción
`open`	Crea un fichero, o abre uno existente
`read`	Lee datos de un fichero
`write`	Escribe datos en un fichero
`close`	Cierra un fichero
`unlink, remove`	Elimina un fichero existente

Crear y abrir ficheros

```
int open (
  const char Ruta,        // Nombre de fichero
  int        Indicadores, // Modo de apertura
  [mode_t    Permisos]    // Permisos en caso de creación
)
```

Devuelve:
- El valor –1 en caso de error
- Un valor entero que representa a un descriptor del fichero creado o abierto

Indicador	Descripción
O_RDONLY	Apertura en modo lectura
O_WRONLY	Apertura en modo escritura
O_RDWR	Apertura en modo lectura/escritura
O_CREAT	Crea el fichero si no existe
O_TRUNC	Trunca el contenido del fichero
O_APPEND	El desplazamiento se sitúa al final del fichero

Crear y abrir ficheros

- Ejemplos

```
int DescriptorDeFichero ;

/*
 * Abrir un fichero en modo sólo lectura
 */
DescriptorDeFichero = open("datos", O_RDONLY) ;

...

/*
 * Abrir un fichero en modo sólo escritura. Si el fichero
 * existe, truncar su tamaño a cero bytes.  Si no existe,
 * crearlo con permisos de lectura y escritura
 */
DescriptorDeFichero = open("datos",
                           O_WRONLY | O_TRUNC | O_CREAT,
                           0666) ;
```

Leer de un fichero

```
ssize_t read (
  int     DescriptorDeFichero,
  void *  Buffer,         // Buffer de lectura
  size_t  NumeroDeBytes  // Numero de bytes a leer
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 si se ha llegado al final del fichero
- Un valor entero que indica el número de caracteres que se han leído

- El primer argumento
  - Indica un descriptor de fichero abierto previamente con open( )
- El segundo argumento
  - Es la dirección de un buffer en el que se almacenarán los bytes leídos
- El tercer argumento
  - Indica el número de bytes a leer

Leer de un fichero

- Ejemplos

```
int   DescriptorDeFichero     ;
char  BufferDeLectura[BUFSIZ] ;
int   BytesLeidos             ;

DescriptorDeFichero = open("/etc/passwd", O_RDONLY) ;

BytesLeidos = read(DescritorDeFichero,
                   BufferDeLectura,
                   BUFSIZ) ;
```

```
int     DescriptorDeFichero ;
int     BufferDeEnteros[50] ;
double  BufferDouble        ;

...
BytesLeidos = read(DescritorDeFichero,
                   (char *)BufferDeEnteros,
                   50*sizeof(int)) ;

BytesLeidos = read(DescriptorDeFichero,
                   (char *)&BufferDouble,
                   sizeof(double)) ;
```

Leer de un fichero

- A considerar
  - El sistema no comprueba si el buffer tiene la capacidad indicada por el tercer argumento
  - El número de bytes devueltos puede ser igual o menor que el indicado por el tercer argumento
  - Cuanto menor sea el tamaño del buffer, mayor número de llamadas hay que efectuar para leer un fichero
  - Existe un valor constante denominado `BUFSIZ`, que indica el tamaño adecuado que debe tener el buffer para leer de ficheros de disco

Escribir en un fichero

```
ssize_t write (
  int     DescriptorDeFichero,
  void *  Buffer,         // Buffer de escritura
  size_t  NumeroDeBytes   // Numero de bytes a escribir
)
```

Devuelve:
- El valor –1 en caso de error
- Un valor entero que indica el número de caracteres que se han escrito

- El primer argumento
  - Indica un descriptor de fichero abierto previamente con open( )
- El segundo argumento
  - Es la dirección de un buffer en el que están almacenados los bytes a escribir
- El tercer argumento
  - Indica el número de bytes a escribir

Escribir en un fichero

- Ejemplos

```
int   DescriptorDeFichero    ;
char  BufferDeEscritura[256] ;
int   BytesEscritos          ;
Int   Contador               ;

DescriptorDeFichero = open("/tmp/prueba.txt", O_WRONLY) ;

for (Contador = 0; Contador ++; Contador < 10)
  BytesEscritos = write(DescriptorDeFichero,
                        (char *)&Contador,
                        sizeof(double)) ;

strcpy(BufferDeEscritura, "#include <stdio.h>" ;
BytesEscritos = write(1,
                      BufferDeEscritura,
                      strlen(BufferDeEscritura)) ;
```

Escribir en un fichero

- `write()` es la llamada complementaria a `read()`
- Al igual que `read()`, hay que considerar
  - El número de bytes devueltos puede ser igual o menor que el indicado por el tercer argumento
  - Cuanto menor sea el tamaño del buffer, mayor número de llamadas hay que efectuar para leer un fichero
  - Existe un valor constante denominado `BUFSIZ`, que indica el tamaño adecuado que debe tener el buffer para escribir en ficheros de disco

Cerrar un fichero

```
int close (
  int DescriptorDeFichero
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta llamada cierra un descriptor de fichero que ha sido abierto previamente
- El sistema cierra todos los descriptores abiertos cuando un proceso acaba su ejecución

Es conveniente cerrar los descriptores explícitamente, cuando no se van a necesitar
- Los descriptores abiertos consumen recursos
- Se evitan algunos errores

Borrar un fichero

```
int unlink (
  const char * Ruta // Nombre del fichero
)

int remove (
  const char * Ruta // Nombre de fichero o directorio
)
```

Devuelven:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Las dos llamadas son equivalentes cuando se borran ficheros
- `remove()` permite también borrar directorios vacíos

Funciones avanzadas

- Operaciones adicionales que se pueden llevar a cabo sobre ficheros

Llamada	Descripción
`link`	Crea un enlace a un fichero
`lseek`	Modifica la posición del puntero de lectura/escritura
`fcntl`	Permite realizar ciertas operaciones con descriptores abiertos
`dup/dup2`	Duplican descriptores abiertos
`stat, fstat`	Devuelven información sobre un fichero
`chmod`	Cambia los permisos de un fichero
`chown`	Cambia el propietario de un fichero

Crear un nuevo enlace

```
int link (
  const char * NombreDeFicheroOriginal,
  const char * NombreDeFicheroNuevo
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- La llamada link()
  - Permite crear un nuevo enlace (de tipo hard)
  - El nombre `NombreDeFicheroNuevo` hará referencia al fichero de nombre `NombreDeFicheroOriginal`

Acceso aleatorio

```
off_t lseek (
  int   DescriptorDeFichero,
  off_t Desplazamiento,
  int   IndicadorDeInicio
)
```

Devuelve:
- El valor –1 en caso de error
- En caso de éxito, el nuevo desplazamiento (*offset*) del puntero de lectura/escritura

- El segundo argumento
  - Indica el número de bytes a desplazar
  - Puede tener un valor negativo
- El tercer argumento
  - Indica desde dónde se desplazará el puntero de lectura/escritura

Crear un nuevo enlace

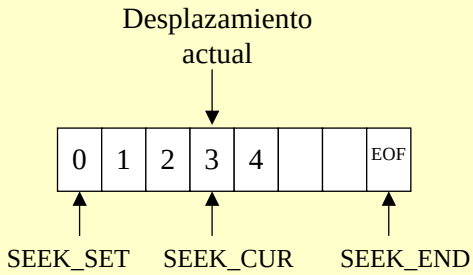
- Operaciones adicionales que se pueden llevar a cabo sobre ficheros

Llamada	Descripción
`link`	Crea un enlace a un fichero
`lseek`	Modifica la posición del puntero de lectura/escritura
`fcntl`	Permite realizar ciertas operaciones con descriptores abiertos
`dup/dup2`	Duplican descriptores abiertos
`stat, fstat`	Devuelven información sobre un fichero
`chmod`	Cambia los permisos de un fichero
`chown`	Cambia el propietario de un fichero

Acceso aleatorio

- El tercer argumento puede tomar los siguientes valores

Indicador	Significado
`SEEK_SET`	El desplazamiento se realiza desde el principio del fichero
`SEEK_CUR`	El desplazamiento se realiza desde la posición actual del puntero de lectura/escritura
`SEEK_END`	El desplazamiento se realiza desde el final del fichero



Acceso aleatorio

- Ejemplos

```
int         DescriptorDeFichero ;
off_t       NuevaPosicion       ;
char [256]  Cadena              ;

...
NuevaPosicion = lseek(DescriptorDeFichero,
                      (off_t)12,
                      SEEK_CUR) ;

NuevaPosicion = lseek(DescriptorDeFichero,
                      (off_t)-25,
                      SEEK_END) ;
...
NuevaPosicion = lseek(DescriptorDeFichero,
                      (off_t)0,
                      SEEK_END) ;
sprintf(Cadena,
        "La longitud del fichero es %s",
        NuevaPosicion) ;

write(1, cadena, strlen(Cadena)) ;
```

Acceso aleatorio

- A considerar
  - Un desplazamiento resultante menor que 0 producirá un error
  - Un desplazamiento resultante mayor que la longitud del fichero dará un error si a continuación se lee
  - Pero la operación válida si es una escritura. El espacio entre la longitud original y la nueva escritura se rellenará con el carácter `NULL`.

Operar sobre descriptores abiertos

```
int fcntl (
  int     DescriptorDeFichero,
  int     Operacion,
  ...                 // número variable de argumentos
)
```

Devuelve:
- El valor –1 en caso de error
- En caso de éxito, depende de la operación

- El segundo argumento
  - Indica la operación a realizar
- El tercer argumento
  - Depende del segundo argumento
- Operaciones
  - Bloqueo de parte (o la totalidad de un fichero), devolver el estado de un descriptor, ...

Duplicar descriptores

```
int dup (
  int     DescriptorDeFichero,
)
```

```
int dup2 (
  int     DescriptorDeFicheroOriginal,
  int     DescriptorDeFicheroNuevo
)
```

Devuelve:
- El valor –1 en caso de error
- El valor del nuevo descriptor

- La llamada `dup()`
  - Busca el primer descriptor libre en la tabla de ficheros y hace que apunte a la misma entrada de la tabla de ficheros que `DescriptorDeFichero`

Duplicar descriptores

- La llamada `dup2()`
  - Cierra `DescriptorDeFicherosNuevo`, si estaba abierto, y hace que apunte a la misma entrada de la tabla de ficheros que `DescriptorDeFicherosAntiguo`
- Tanto `dup()` como `dup2()`
  - Son llamadas que se utilizan para redireccionar la entrada y la salida de un proceso

Devolver información de un fichero

```
int stat (
  const char * Ruta    // Nombre del fichero,
  struct stat* Buffer
)
```

```
int fstat (
  int          DescriptorDeFicheroOriginal,
  struct stat* Buffer
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Ambas llamadas
  - Devuelven en la estructura `Buffer` información sobre un fichero
- El contenido de la estructura `stat`
  - Está definido en `<sys/types.h>`

Devolver información de un fichero

- La estructura `stat` contiene, entre otros campos

Campo	Descripción
`st_dev`	Dispositivo lógico en el que está almacenado el fichero
`st_ino`	Número de nodo índice
`st_mode`	Máscara de modo
`st_nlink`	Número de enlaces no simbólicos
`st_uid`, `st_gid`	Identificadores de usuario y grupo del fichero
`st_size`	Tamaño lógico del fichero, en bytes
`st_atime`, `st_mtime`, `st_ctime`	Fechas de último accesso, última modificación y última modificación de información de tipo administrativo
`st_blksize`	Tamaño de bloque
`st_blocks`	Número de bloques físicos del fichero

Cambiar los permisos

```
int chmod (
  const char * Ruta       // Nombre del fichero
  mode_t       NuevoModo
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta llamada
  - Permite modificar los permisos de un fichero

Cambiar el propietario

```
int chown (
  const char * Ruta        // Nombre del fichero,
  uid_t        IdentificadorDePropietario
  gid_t        IdentificadorDeGrupo
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta llamada
  - Permite cambiar el propietario y el grupo de un fichero

Llamadas al sistema: directorios

- Operaciones básicas que se pueden realizar sobre directorios

Operación	Descripción
`mkdir`	Crea un directorio
`rmdir`	Borra un directorio vacío
`opendir`	Abre un directorio
`closedir`	Cierra un directorio
`readdir`	Lee una entrada de directorio
`rewindir`	Posiciona el puntero de lectura en la primera entrada
`seekdir`	Posiciona el puntero de lectura de forma aleatoria
`telldir`	Devuelve la posición del puntero
`chdir`	Cambia el directorio de trabajo
`getcwd`	Devuelve el directorio de trabajo

Crear un directorio

```
int mkdir (
  const char * Ruta       // Nombre del directorio
  mode_t       Modo
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta llamada
  - Permite crear un directorio vacío

Borrar un directorio

```
int rmdir (
  const char * Ruta       // Nombre del directorio
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta llamada
  - Permite borrar un directorio siempre y cuando esté vacío

Abrir un directorio

```
DIR* opendir (
  const char * Ruta       // Nombre del directorio
)
```

Devuelve:
- El valor NULL en caso de error
- Un puntero a una estructura `DIR` en caso de éxito

- Esta función
  - Permite abrir un directorio
- Observaciones
  - El puntero `DIR` apunta a la primera entrada del directorio
  - La estructura `DIR` se maneja de forma parecida a la estructura `FILE` de la biblioteca estándar de C

Cerrar un directorio

```
int closedir (
  DIR * PunteroADirectorio
)
```

Devuelve:
- El valor -1 en caso de error
- El valor 0 en caso de éxito

- Esta función
  - Permite cerrar un directorio

Leer un directorio

```
struct dirent * readdir (
  DIR * PunteroADirectorio
)
```

Devuelve:
- En caso de éxito, un puntero a una entrada de directorio
- El valor NULL si se llega al final del directorio

- Esta función
  - Devuelve un puntero a una estructura `dirent`
  - Avanza el puntero de lectura del directorio a la siguiente entrada

Control del puntero de lectura

```
void rewinddir (
  DIR * PunteroADirectorio
)
```

Devuelve: Nada

- Esta función
  - Sitúa el puntero de lectura en la primera entrada del directorio

Control del puntero de lectura

```
void seekdir (
  DIR * PunteroADirectorio
  long  Posicion
)
```

Devuelve: Nada

- Esta función
  - Sitúa el puntero de lectura en la posición indicada por el segundo argumento

Control del puntero de lectura

```
long telldir (
  DIR * PunteroADirectorio
)
```

Devuelve:
- El valor –1 en caso de error
- La posición del puntero de lectura, en caso de éxito

- Esta función
  - Devuelve la posición del puntero de lectura

Cambiar el directorio de trabajo

```
int chdir (
   const char * Ruta // Nombre del directorio
)
```

Devuelve:
- El valor –1 en caso de error
- El valor 0 en caso de éxito

- Esta función
  - Cambia el directorio de trabajo a `Ruta`

Obtener el directorio de trabajo

```
char * getcwd (
  char * NombreDelDirectorio
  size_t Longitud  // Longitud de NombreDelDirectorio
)
```

Devuelve:
- En caso de éxito, un puntero a nombre del directorio de trabajo
- El valor `NULL` en caso de fallo

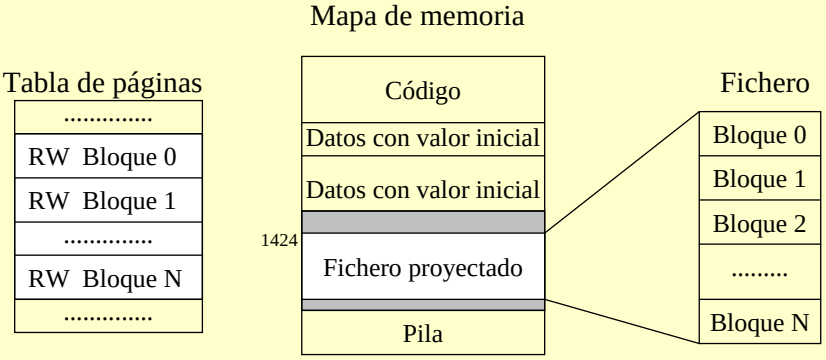
- Esta función
  - En caso de éxito, el nombre del directorio de trabajo es copiado al array pasado como primer argumento

Aspectos avanzados

- Ficheros proyectados en memoria
- Sistemas de ficheros estructurados en logs
- Sistemas de ficheros multimedia

Ficheros proyectados en memoria

- Mediante memoria virtual
  - Se hace corresponder entradas en la tabla de páginas con bloque de ficheros ejecutables
- Ficheros proyectados en memoria
  - Se usa la misma técnica pero sobre cualquier tipo de fichero



Primitivas POSIX

```
caddr_t mmap (
  caddr_t Direccion    // Direccion del mapa
  size_t  longitud     // tamaño a proyectar
  int     proteccion
  int     indicador
  int     descriptor   // desc. del fichero a proyectar
  off_t   desplazamiento
)
```

Devuelve:
- El valor –1 en caso de error ??????????
- El valor 0 en caso de éxito??????????

- Esta función
  - Proyecta un fichero
- El primer argumento
  - Si es cero, el sistema decide dónde proyectar el fichero

Primitivas POSIX

- Argumentos
  - `Longitud` indica el tamaño de la zona a proyectar
  - `Protección`
    - `PROT_READ`
    - `PROT_WRITE`
    - `PROT_EXEC`
  - `Indicador`
    - `MAP_SHARED`
    - `MAP_PRIVATE`
  - `Desplamiento` indica a partir de qué byte del fichero se produce la proyección

Primitivas POSIX

```
caddr_t munmap (
  caddr_t Direccion    // Direccion del mapa
  size_t  longitud     // tamaño a desproyectar
)
```

Devuelve:
- El valor –1 en caso de error ???????????
- El valor 0 en caso de éxito??????????

- Esta función
  - Deshace una proyección, parcial o totalmente

Ejemplo: contar caracteres de un fichero

```
...
/* PASO 1. Apertura del fichero en modo lectura */
descriptorFichero = open(argv[2], O_RDONLY) ;

/* PASO 2. Obtener la longitud del fichero */
fstat(descriptorFichero, &informacionFichero) ;
longitud = informacionfichero.st_size ;

/* PASO 3. Proyeccion del fichero en memoria */
ptr = mmap((caddr_t) 0, longitud, PROT_READ, MAP_SHARED,
           descriptorFichero,0) ;

/* PASO 4. Cierre del fichero */
close(descriptorFichero) ;

/* PASO 5. Contar los caracteres */
inicio_ptr = ptr ;
for (i = 0; i < longitud; i++)
  if (*ptr++ == caracter) contador ++ ;

/* PASO 6. Deshacer la proyeccion */
munmap(inicio_ptr, longitud) ;
...
```

Ejemplo: copiar dos ficheros

```
...
/* PASO 1. Apertura del fichero origen */
ficheroOrigen_df = open(argv[2], O_RDONLY) ;

/* PASO 2. Crear el fichero destino */
ficheroDestino_df = open(argv[2],
                         O_CREAT | O_TRUNC | O_RDWR,
                         0640) ;

/* PASO 3. Obtener la longitud del fichero origen */
fstat(ficheroOrigen_df, &informacionFicheroOrigen)
longitud = informacionFicheroOrigen.st_size ;

/* PASO 4. Establecer la longituddel fichero destino */
ftruncate(ficheroDestino_df, longitud) ;

/* PASO 5. Proyeccion del fichero origen en memoria */
ptr1 = mmap((caddr_t) 0, longitud, PROT_READ, MAP_SHARED,
            ficheroOrigen_df, 0) ;

/* PASO 6. Proyeccion del fichero destino en memoria */
ptr2 = mmap((caddr_t) 0, longitud, PROT_WRITE,MAP_SHARED,
            ficheroDestino_df, 0) ;
```

Ejemplo: copiar dos ficheros

```
...

/* PASO 7. Cierre de los ficheros */
close(ficheroOrigen_df) ;
close(ficheroDestino_df) ;

/* PASO 8. Copia de los ficheros */
zona1_ptr = ptr1 ;
zona2_ptr = ptr2 ;
for (i = 0; i < longitud; i++) {
  *ptr2++ = *ptr1++ ;
} /* for */

/* PASO 9. Deshacer las proyecciones */
munmap(zona1_ptr, longitud) ;
munmap(zona2_ptr, longitud) ;

...
```

Sistemas de ficheros estructurados en logs

- Los sistemas de ficheros estructurados en logs (*Log-structured File Systems* o LFSs) se basan en
  - Los procesadores son cada vez más rápidos
  - La memoria principal cada vez es mayor
- Consecuencia
  - Se pueden satisfacer la mayor parte de las operaciones de lectura sin tener que acceder a disco
- Luego
  - La mayor parte de las operaciones sobre disco serán de escritura
  - Las técnicas de lectura adelantada no serán de utilidad

Sistemas de ficheros estructurados en logs

- En la actualidad
  - La mayor parte de las escrituras escriben pocos datos, por lo que no se amortizan los tiempos de búsqueda y de latencia
- Ejemplo en UNIX: creación de un fichero
  - Hay que modificar
    - El nodo índice del directorio
    - El bloque de datos que tiene el directorio
    - El nodo índice del fichero
    - Los bloques de datos del fichero
  - Se podrían retener las modificaciones en memoria
    - Pero existe el riesgo de inconsistencias
    - Luego los nodos índice suelen ser escritos a disco inmediatamente

Sistemas de ficheros estructurados en logs

- El objetivo de un LFS es
  - Sacar el máximo provecho a la velocidad de los discos, aunque se realicen pocas escrituras y éstas sean aleatorias
  - Para ello es necesario modificar el esquema de implementación clásico de los sistemas de ficheros en UNIX
- Idea básica
  - Estructurar el disco como un log

Sistemas de ficheros estructurados en logs

- Funcionamiento
  - Periódicamente, todas las escrituras pendientes que están en memoria se agrupan y se escriben en un único segmento contiguo al final del log
  - Un segmento contiene una mezcla de nodos índice, bloques de datos y bloques de directorios
  - Al inicio de cada segmento hay información sobre su contenido
  - Se busca que el tamaño medio sea de 1MB, lo que parece aprovechar al máximo la velocidad del disco

Sistemas de ficheros estructurados en logs

- Los nodos índice
  - Tienen la misma estructura y funcionan igual que en un sistema UNIX tradicional
  - Pero ahora están repartidos entre los segmentos del log, en lugar de una zona concreta del disco
  - Para localizarlos existe un mapa indexado por el número de nodo índice. Este mapa se copia en memoria para acelerar las búsquedas
- Problemas cuando el disco se llena
  - Una hebra recorre periódicamente el log, compactándolo
- Rendimiento respecto al sistema clásico
  - Mejoras en un orden de magnitud para escrituras pequeñas, y similar en escrituras grandes

Sistemas de ficheros multimedia

- El uso de información multimedia (vídeo, audio) es cada vez más común
- Esta información tiene dos características clave
  - Hay que procesar cantidades ingentes de datos
    - Por ejemplo, un fichero de audio que contenga tres o cuatro minutos de música puede ocupar en torno a 50 MB sin aplicar técnicas de compresión
  - Hay que cumplir restricciones de tiempo real (calidad de servicio)
- Un SO multimedia difiere de uno convencional en
  - Planificación de procesos, sistema de ficheros y planificación de discos

Sistemas de ficheros multimedia

- Un sistema de ficheros tradicional se amolda a un esquema *pull server*
  - El usuario ha de solicitar bloques de datos invocando repetidamente una operación de lectura para obtener los datos unos detrás de otros
- Un sistema de ficheros multimedia se amolda a un esquema *push server*
  - El usuario invoca una operación de inicio de transmisión sobre un servidor,
  - El servidor responde enviando trozos de información de forma continuada, a la frecuencia necesaria para satisfacer los requisitos de calidad de servicio establecidos mediante parámetros de la operación de inicio

Sistemas de ficheros multimedia

- La mayoría de los ficheros multimedia
  - Son de gran tamaño
  - Se escriben una vez
  - Se leen muchas veces
  - El acceso secuencial es el más usado
- Aspectos a considerar
  - Utilizar asignación contigua para almacenar los ficheros
  - Usar varios discos en paralelo para aumentar el rendimiento (por ejemplo, RAID)
  - Aprovechar que el acceso a la información multimedia es predecible para aplicar técnicas de planificación de discos eficientes

Recursos en la Web

- Bibliografía
- Recursos en la Web

Bibliografía

- [Tanenbaum, 2001]
  - Andrew. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall. 2001.
- [Nutt, 2000]
  - Gary Nutt. *Operating Systems. A Modern Perspective. 2nd Edition*. Addison-Wesley. 2000.
- [Stallins, 2001]
  - William Stallins. *Operating Systems. 4th Edition*. Prentice-Hall. 2001.
- [Carretero et al, 2001]
  - J. Carretero, F. García, P. De Miguel, F. Pérez. *Sistemas Operativos. Una visión aplicada*. McGraw Hill. 2001

Bibliografía

- [SG, 1998]
  - A. Silberschatz, P. Galvin. *Operating System Concepts. 5th Edition*. Addison-Wesley. 1998.

Recursos en la Web