SISTEMAS OPERATIVOS

Grupo 2 sistem.operativos@gmail.com

5. Sistema de Archivos

Objetivo:

El alumno describirá las diferentes formas de organización y acceso a archivos, basándose en el modelo de sistema de archivos.

Introducción

Desde el punto de vista de los usuarios y las aplicaciones, los archivos y directorios son los elementos centrales del sistema. Cualquier usuario genera y usa información a través de las aplicaciones que ejecuta en el sistema. En todos los sistemas operativos de propósito general, las aplicaciones y sus datos se almacenan en archivos ...

Introducción

... no volátiles, lo que permite su posterior reutilización.

Los usuarios ven los archivos como un conjunto de información estructurada según sus necesidades o las de sus aplicaciones, mientras que el sistema operativo los contempla como conjuntos estructurados según sus necesidades de almacenamiento y representación.

Introducción

- Cualquiera que sea la visión de los archivos, para los usuarios su característica principal es que no están ligados al ciclo de vida de una aplicación en particular.

Introducción

- El servidor de archivos es la parte del sistema operativo que se ocupa de facilitar el manejo de los dispositivos periféricos, ofreciendo una visión lógica simplificada de los mismos en forma de archivos.

Introducción

- Mediante esta visión lógica se ofrece a los usuarios un mecanismo de abstracción que oculta todos los detalles relacionados con el almacenamiento y distribución de la información en los dispositivos periféricos, así como el funcionamiento de los mismos.

Introducción

- Todas las aplicaciones en una computadora necesitan almacenar y recuperar información cumpliendo lo siguiente:
- Superando las limitaciones inherentes al almacenamiento

Introducción

- Trascendiendo a la duración de los procesos que las utilizan o generan.
- Independizando a la información de los procesos permitiendo el acceso a la misma a través de varios procesos.

Introducción

Las condiciones esenciales para el almacenamiento de la información a largo plazo son:

- Debe ser posible almacenar una cantidad muy grande de información.

Introducción

- La información debe sobrevivir a la conclusión del proceso que la utiliza.
- Debe ser posible que varios procesos tengan acceso concurrente a la información.

Archivo

Todos los sistema operativos proporcionan una unidad de almacenamiento lógico que oculta los detalles del sistema físico de almacenamiento, llamada **archivo**

Archivo

- *Un archivo es una unidad de almacenamiento lógico no volátil que agrupa un conjunto de información relacionada entre sí bajo un mismo nombre.*

Archivo

Para el usuario los archivos son contenedores de información de un tipo definido por su creador, aunque todos ellos se pueden agrupar en dos grandes clases: **archivos ASCII y archivos binarios.**

Archivo

Los archivos ASCII pueden ser formados por líneas de texto, pueden ser editados o impresos directamente cosa que no suele ocurrir con los archivos binarios que suelen almacenar archivos ejecutables, objetos y datos no textuales.

Archivo

En el Sistema Operativo UNIX existen diferentes tipos de archivos:

- Ordinario
- Directorio
- Caracter

Archivo

- Bloque
- Liga
- Socket
- Fifo o pipe

Tarea: Investigar sobre cada uno de ellos y ejemplificar mediante la creación de cada uno de los diferentes tipos.

Archivo

Desde el punto de vista del sistema operativo un archivo se caracteriza por tener una serie de atributos. Dichos atributos varían de unos sistemas operativos a otros, pero habitualmente incluyen los siguientes:

Archivo

- Nombre: identificador del archivo en formato comprensible para el usuario. Definido por su creador.
- Identificador único: en el ámbito interno, el sistema operativo no usa el nombre para identificar los archivos, sino un identificador único, este identificador suele ser un número y desconocido para los usuarios

Archivo

- Tipo de archivo: útil en aquellos sistemas operativos que soportan tipos de archivos en el ámbito interno. En algunos casos el tipo de archivo se identifica por el llamado número mágico, el cual es una etiqueta del sistema operativo que le permite distinguir entre distintos formatos de almacenamiento de archivos.

Archivo

- Mapa del archivo: formado normalmente por apuntadores a los dispositivos y a los bloques dentro de estos, que guardan el archivo. Esta información se utiliza para localizar el dispositivo y los bloques donde se almacena la información que contiene el archivo.

Archivo

- Protección: información de control de acceso que define quien puede hacer que sobre el archivo, la palabra clave para acceder al archivo, el dueño del archivo, su creador, etc.

Archivo

- Tamaño del archivo: número de bytes en el archivo, máximo tamaño posible para el archivo, etc.
- Información temporal: tiempo de creación, de último acceso, de última actualización, etc.

Archivo

- Información de control del archivo: que indica si es un archivo oculto, de sistema, normal o directorio, atributos, etc.

Tablas descriptoras de archivos

- El **nodo-i de UNIX** contiene información acerca del propietario del archivo, de su grupo del modo de protección aplicable al archivo, el número de enlaces al archivo, de fechas de creación de actualización, el tamaño y el tipo del mismo

Tablas descriptoras de archivos

- Incluye un mapa del archivo mediante apuntadores a bloques del dispositivo que contiene datos del archivo. Cuando se abre un archivo su nodo-i se trae a memoria.

Tablas descriptoras de archivos

- **Registro MFT** (Master File Table), fue introducido desde Windows NT y a continuado en sus sucesores (2000, XP, Server 2003 y Vista). Describe el archivo mediante los siguientes atributos: cabecera, información estándar, descriptor de seguridad, nombre de archivo,

Tablas descriptoras de archivos

y datos. A diferencia del nodo-i de UNIX un registro MFT permite almacenar hasta 1.5 KB de datos del archivo en el propio registro, si el archivo es mayor dentro del registro se almacena información del mapa físico del archivo incluyendo punteros a grupos de bloques de datos.

Tablas descriptoras de archivos

En el caso de MS-DOS tenemos entradas de directorio. La representación del archivo es bastante sencilla debido principalmente por ser un sistema monoproceso y monousuario

Tablas descriptoras de archivos

En este caso la información de protección no existe limitandose a unos atributos minimos que permiten ocultar el archivo o ponerlo como de solo lectura, en la descripción se incluye el nombre y el tamaño en KB, además se especifíca la posición del inicio del archivo en la tabla FAT (file allocation table), donde se almacena el mapa físico del archivo.

Nombres de archivo

Una de las características principales de un sistema operativo de cara a los usuarios es la forma en que se nombran los archivos. Todo objeto archivo debe tener un nombre a través del que se pueda acceder a él de forma inequívoca.

Nombres de archivo

El tipo de nombres que se usan para los archivos varía de un sistema operativo a otro. Sin embargo y para ser mas amigables con los usuarios todos ellos permiten asignar a los archivos nombres formados por combinaciones de caracteres alfanuméricos y algunos caracteres especiales.

Nombres de archivo

Por ejemplo, MS-DOS permite nombres con una longitud máxima de 8 caracteres, mientras UNIX permite nombres de hasta 4096 caracteres. Asimismo algunos sistemas como MS-DOS o Windows no distinguen entre mayúsculas y minúsculas, mientras otros como UNIX sí lo hacen.

Nombres de archivo

Muchos sistemas operativos permiten añadir una o más extensiones al nombre de un archivo. Dichas extensiones sirven para indicar al sistema operativo, a las aplicaciones o a los usuarios características del contenido del archivo.

Nombres de archivo

En MS-DOS, por ejemplo un archivo solo puede tener un extensión y esta debe ser una etiqueta de tres letras como máximo. En UNIX y los Windows modernos un archivo pueden tener cualquier número de extensiones y de cualquier tamaño.

Nombres de archivo

Tipo de archivo	Extensión usual	Función
Ejecutable	exe, com, bin o ninguna	programa en lenguaje de máquina listo para ejecutarse
Objeto	obj, o	compilado, en lenguaje de máquina, no enlazado
Código fuente	c, p, pas, f77, asm, a	código fuente en diversos lenguajes
Por lotes	bat, sh	órdenes al intérprete de órdenes
Texto	txt, doc	datos textuales, documentos
Procesador de textos	wp, tex, rrf, etc	formatos de diversos procesadores de textos
Biblioteca	lib, a	bibliotecas de rutinas para programadores
Impresión o visualización	ps, dvi, gif	archivo en ASCII o binario en un formato para impresión o visualización
Archivado	arc, zip, tar	archivos relacionados agrupados en un solo archivo, a veces comprimido para almacenarse

Nombres de archivo

Habitualmente las extensiones son significativas sólo para las aplicaciones de usuario, pero existen casos en que el sistema operativo reconoce y da soporte específico, a distintos tipos de archivos según sus extensiones, ambo enfoques tienen ventajas y desventajas.

Nombres de archivo

- Si el sistema operativo reconoce tipos de archivos, puede proporcionar utilidades específicas y muy optimizadas para los mismos. Sin embargo el diseño del sistema se complica mucho.

Nombres de archivo

- Si no los reconoce, las aplicaciones deben programar las utilidades específicas para manejar su tipo de archivos. Pero el diseño interno del sistema operativo es mucho más sencillos.

Nombres de archivo

Ejemplo de la primera opción fue el sistema operativo TOPS-20, que proporcionaba distintos tipos de archivo en su ámbito interno y ejecutaba operaciones de forma automática dependiendo del tipo de archivo.

Nombres de archivo

UNIX es un sistema operativo para el que las extensiones del nombre no son significativas, este sistema proporciona un único tipo de archivo cuyo modelo se basa en una tira secuencial de bytes.

Nombres de archivo

Windows tampoco es sensible a las extensiones de archivos, pero sobre el existen aplicaciones como el explorador que permiten asociar dichas extensiones con la ejecución de aplicaciones.

Estructura de un archivo

Desde el punto de vista del sistema operativo algunos archivos como los ejecutables o las bibliotecas dinámicas deben tener cierta estructura para que su información pueda ser interpretada.

Estructura de un archivo

Desde el punto de vista del usuario, la información de un archivo puede estructurarse como una lista de caracteres, un conjunto de registros secuencial o indexado etc.

Estructura de un archivo

Windows y UNIX proporcionan una estructura interna de archivo e interfaz de programación muy sencillos permitiendo a las aplicaciones construir cualquier tipo de estructura para sus archivos sin que el sistema operativo sea consciente de ello.

Estructura de un archivo

Todos los sistema operativos deben de reconocer al menos un tipo de archivo: sus propios ejecutables. La estructura de un archivo ejecutable está íntimamente ligada a la forma de gestionar la memoria y la E/S en un sistema operativo.

Estructura de un archivo

El algunos se reconocen por su extensión (VMS), en UNIX por ejemplo con el número mágico.

Operaciones con Archivos

- ***Crear un archivo:***
  - •1º Encontrar espacio para él en el sistema de archivos.
  - •2º Insertar una entrada para el nuevo archivo en el directorio (esta entrada registra el nombre del archivo y su ubicación en el sistema de archivos).
- ***Eliminar un archivo:***
  - • Libera todo el espacio que el archivo ocupa.
  - • Se borra la entrada del directorio.
- ***Escritura de un archivo:***
  - •Especificar el nombre del archivo y la información que se escribirá en él.
  - •Debemos mantener un apuntador de escritura (que se irá actualizando cada vez que se escriba en el archivo) que indique en qué posición del archivo se efectuará la siguiente escritura. Dicho apuntador deberá actualizarse cada vez que se escriba en el archivo.

Operaciones con Archivos

Leer un archivo:

- Especificar nombre y lugar (de la memoria) donde colocar el siguiente bloque del archivo.
- Se necesita un apuntador para la lectura a la posición del archivo donde se efectuará la siguiente lectura.
- Nota: En general, un archivo o bien se lee o se escribe con lo que la mayor parte de los sistemas operativos sólo mantienen un apuntador a la posición actual en el archivo. Ambas operaciones usan el mismo apuntador, ahorrando espacio y aportando sencillez.

Reubicación dentro de un archivo:

- Búsqueda en un archivo.

Truncar un archivo:

- Borrar el contenido de un archivo pero haciendo que sus atributos no cambien (salvo el tamaño del archivo).

Otras Operaciones Secundarias:

- ***Appending***: Añadir información nueva al final de un archivo ya existente.
- ***Rename***: Renombrado de un archivo.

Operaciones con Archivos

Las llamadas al sistema más comunes relacionadas con los archivos se enumeran a continuación:

1. **CREATE** (Crea un archivo).
2. **DELETE** (Elimina el archivo del dispositivo de almacenamiento).
3. **OPEN** (Conecta el archivo al proceso).
4. **CLOSE** (Desconecta el archivo del proceso).
5. **READ** (Lee información del archivo).
6. **WRITE** (Escribe información en el archivo).
7. **APPEND** (Escribe información al final).
8. **SEEK** (Se posiciona en un archivo de acceso aleatorio).

Métodos de acceso

Acceso Secuencial:

- Método de acceso más común.
- La información del archivo se procesa en orden, un registro tras otro.
- El archivo debe tener la capacidad de adelantarse secuencialmente y también de restablecerse al principio.
- El apuntador del archivo apunta a la posición que se acaba de leer y escribir.

- **Desventaja:**
  - Problema de inserción de un bloque.
- **Ventaja:**
  - Aprovechamiento del espacio cuando no se necesita hacer modificaciones.
- **Nota:** El sistema es bueno para hacer backups.
- **Primitivas:** *write(datos), read(datos), rewind.*

Métodos de acceso

Acceso Directo:

- **Archivo:** Secuencia numerada de registros lógicos de longitud fija.
- Permite que podamos leer y escribir registros rápidamente sin ningún orden específico.
- Útiles para acceder inmediatamente a grandes cantidades de información.
- Se puede especificar la posición del dispositivo en la que se desea leer o escribir.

Primitivas: *write(posición, datos), read(posición, datos), seek(posición).*

Ejemplo: Discos magnéticos.

Métodos de acceso

- **PILA**
- Es la forma más fácil de organizar un archivo. Los datos se recogen en el orden en que llegan.
- Su objetivo es simplemente acumular una masa de datos y guardarla.
- Los registros pueden tener campos diferentes o similares en un orden distinto. Cada campo debe ser autodescriptivo, incluyendo tanto un campo de nombre como el valor. La longitud de cada campo debe indicarse implícitamente con delimitadores, explícitamente incluidos como un subcampo más.

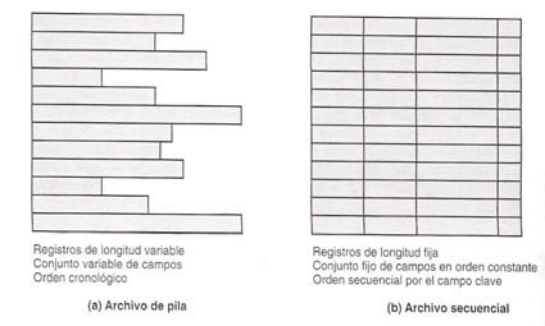
Métodos de acceso

- **PILA**
- El acceso a los registros se hace por búsquedas exhaustiva y son fáciles de actualizar. Si se quiere encontrar un registro que contiene un campo particular y un valor determinado, es necesario examinar cada registro de la pila hasta encontrar el registro deseado. Si se quieren encontrar todos los registros que contienen un campo particular o que tienen un valor determinado para ese campo, debe buscarse el archivo entero.

Métodos de acceso

- **PILA**
- Se aplica cuando los datos se recogen o almacenan antes de procesarlos o cuando no son fáciles de organizar. Esta clase de archivo aprovecha bien el espacio cuando los datos almacenados varían en tamaño y estructura. Fuera de estos usos limitados, este tipo de archivos no se adapta a la mayoría de las aplicaciones.

Métodos de acceso



Métodos de acceso

- **ARCHIVO INDEXADO**
- A los registros se accede solo a través de sus índices. No hay restricción en la ubicación de los registros, al menos un índice contiene un apuntador a cada registro y pueden emplearse registros de longitud variable.
- Se suelen utilizar dos tipos de índices, uno exhaustivo que contiene una entrada para cada registro del archivo principal y se organiza como un archivo secuencial para facilitar la búsqueda,

Métodos de acceso

- .... el otro índice es parcial que contiene entrada a los registros donde esté el campo de interés.
- Con registro de longitud variable, algunos registros no contendrán todos los campos y cuando se añade un registro al archivo principal, todos los archivos de índices deben actualizarse.

Directorios

- Para poder acceder a los archivos con facilidad, todos los sistemas operativos proporcionan formas de organizar los nombres de archivos mediante directorios. Un directorio puede almacenar otros atributos de un archivo tales como ubicación, propietario, etc., dependiendo de cómo haya sido diseñado.

Directorios

- Habitualmente los directorio se implementan como archivos pero se tratan de forma especial y existen servicios especiales del sistema operativo para su manipulación.
- Un directorio es un objeto que relaciona de forma unívoca el nombre de un archivo y el descriptor interno del mismo usado por el sistema operativo.

Directorios

- Los directorios sirven para organizar y proporcionar información acerca de la estructuración de los archivos en el sistema de archivos.
- Habitualmente un directorio contiene tantas entradas como archivos son accesibles a través de él, siendo la función principal de los directorios presentar una visión lógica simple al usuario, escondiendo los detalles de implementación.

Directorios

- Cuando se abre un archivo, el sistema operativo busca en el sistema de directorios hasta que encuentra la entrada correspondiente al nombre del archivo. A partir de dicha entrada el sistema operativo obtiene el identificador interno del archivo y posiblemente algunos atributos del mismo.

Directorios

- Al igual que un archivo un directorio es un objeto y debe ser representado por una estructura de datos. Una cuestión importante del diseño es decidir que información deberá ir asociada a una entrada de directorio.

Para esto tenemos dos alternativas principales:

Directorios

- * Almacenar los atributos del archivo en su entrada de directorio.
- * Almacenar únicamente el identificador del descriptor de archivo y colocar los atributos del objeto archivo dentro de la estructura de datos asociada a su descriptor.

Directorios

- En el diseño de sistemas operativos UNIX se adoptó la segunda alternativa por lo que la entrada de directorio es una estructura de datos muy sencilla, que contiene únicamente el nombre del archivo asociado a ella y el identificador del descriptor de archivo, número nodo-i, usado por el sistema operativo.

Directorios

- El uso de una entrada de directorio como la de UNIX tiene varias ventajas:
- - La entrada de directorio no se ve afectada por los cambios de atributos del archivo.
- - Los nodos-i pueden representar a su vez directorios o archivos, con los que se puede construir esquemas de nombres jerárquicos de forma sencilla.

Directorios

- - La longitud de los nombres no está predeterminada, pudiendo ser variable hasta un límite grande (4,096 caracteres en las versiones actuales).
- - La interpretación de nombres es muy regular, lo que aporta sencillez al sistema de archivos.
- - Facilita la creación de sinónimos para el nombre de archivos.

Elementos de información de un Directorio

- ***Información básica***
- **Nombre del archivo**: nombre elegido por el creador (usuario o programa). Debe ser único en un directorio específico.
- **Tipo de archivo**: por ejemplo: texto, binario, módulo de carga, etc.
- **Organización de archivo**: para sistemas que soportan varias organizaciones.

Operaciones sobre directorios

- ***Buscar:*** cuando un usuario o aplicación hace referencia a un archivo, debe buscarse en el directorio la entrada correspondiente al archivo.
- ***Crear archivo:*** al crear un nuevo archivo, debe añadirse una entrada al directorio.
- ***Borrar archivo:*** al borrar un archivo, debe eliminarse una entrada del directorio.

Operaciones sobre directorios

- ***Listar un directorio:*** puede solicitarse todo el directorio o una parte. Generalmente, esta petición la hace un usuario y el resultado es una lista de todos los archivos poseídos por dicho usuario, junto a algunos de los atributos de cada archivo (tipo, información de control de acceso; información de uso).

Operaciones sobre directorios

Renombrar un archivo: Se puede cambiar el nombre si cambia el contenido o uso del archivo. También puede permitir modificar su posición dentro de la estructura de directorios

Operaciones sobre directorios

- ***Actualizar directorio***: cuando algunos atributos del archivo se almacenan en el directorio, un cambio en alguno de estos atributos requiere un cambio en la entrada del directorio correspondiente.

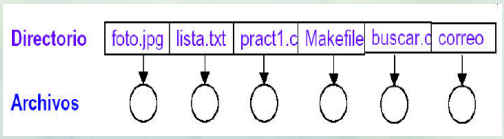
Tipos de organización

- ***Directorio de un solo nivel***:
  - Todos los archivos se guardan en un solo directorio.
  - Estructura más simple y más fácil de soportar
  - ***Características***:
    - Tiene limitaciones importantes cuando el número de archivos aumenta o si hay más de un usuario.

Tipos de organización

- No permite clasificar la información de ninguna manera.
- Como todos los archivos están en el mismo directorio, deben tener nombres únicos.
- Problema: En sistemas multiusuario hay problemas para nombrar los archivos, es decir, puede haber confusión de nombres de archivo entre diferentes usuarios.

Tipos de organización



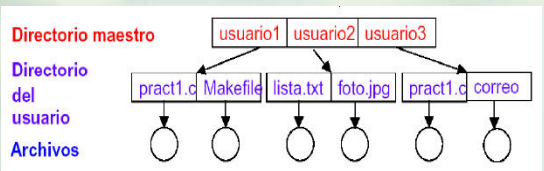
Tipos de organización

- ***Directorio de dos niveles***:
  - Cada usuario tiene su propio directorio de archivos de usuario (UFD, *user file directory*)
  - Características:
    - Se asigna un directorio a cada usuario o tipo de archivo.
    - Existe un directorio maestro o padre (MFD, *master file directory*), por encima.

Tipos de organización

- Las operaciones de un usuario sobre los archivos están restringidas a su dirección.
- Existen operaciones de actualización del directorio maestro.
- Existen problemas para "cooperar" entre usuarios.
- Conceptos: "ruta de acceso" (*pathname*) y "camino de búsqueda" (*path*).
- Fácil administración en cuanto a la protección

Tipos de organización



Estructura de árbol

A medida que la capacidad de los dispositivos se fue incrementando, fueron creciendo también el número de archivos almacenados por los usuarios en el sistema de archivos. Con lo que el problema de la complejidad del nombrado, aunado a la estructura de dos niveles volvió a aparecer.

Estructura de árbol

Fue necesario generalizar la estructura de dos niveles para obtener una estructura jerárquica más general que permitiera a los usuarios ordenar sus archivos con criterios lógicos en sus propios subdirectorios, sin depender de las limitaciones de los niveles de la estructura de directorios.

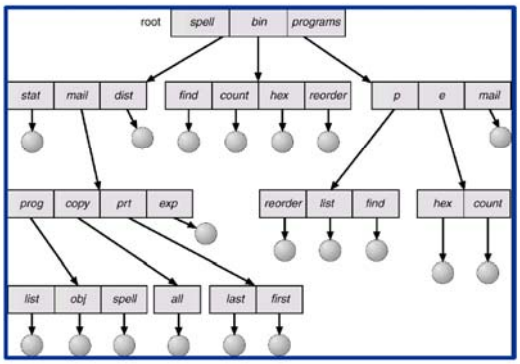
Estructura de árbol

Una estructura de árbol representa todos los directorios y subdirectorios del sistema partiendo de un directorio raíz, existiendo un camino único (path) que atraviesa el árbol desde la raíz hasta un archivo determinado. Los nodos del árbol son directorios que contienen un conjunto de subdirectorios o archivos.

Estructura de árbol

Las hojas del árbol son siempre archivos. Normalmente cada usuario tiene su propio directorio home a partir del cual se cuelgan sus subdirectorios y archivos y en el que le sitúa el sistema operativo cuando entra a su cuenta.

Directorio estructurado en árbol



Grafo acíclico

La estructura de árbol es muy general pero no proporciona los servicios requeridos en algunos entornos.

No existe la forma de compartir archivos o directorios ya que en la estructura de árbol exige que un archivo se llegue con solo una ruta y un nombre.

Grafo acíclico

Sería interesante permitir a los usuarios compartir archivos o subdirectorios llegando a los mismos por sus respectivos directorios de usuario para no tener problemas de seguridad y protección.

Grafo acíclico

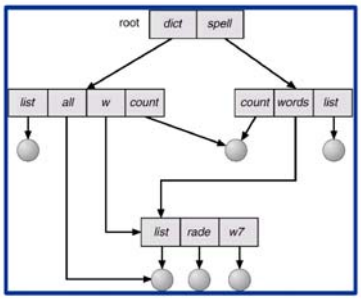
El modelo descrito rompe la relación uno a uno entre el nombre y el archivo, al requerir que un mismo archivo pueda ser accedido a través de varios caminos. Este problema puede resolverse generalizando la estructura de árbol de directorio para convertirla en un grafo acíclico.

Grafo acíclico

En el cual el mismo archivo o subdirectorio puede estar en dos directorios distintos, estableciendo una relación unívoca nombre-archivo. La forma habitual de compartir archivos es crear un enlace o líga al archivo compartido.

Grafo de directorios acíclico

- Tiene ficheros y subdirectorios compartidos.



Grafo acíclico

Enlace físico o duro: Un apuntador a otro archivo cuya entrada de directorio tiene el mismo descriptor de archivo (en UNIX i-nodo) que el archivo enlazado. Ambos nombres apuntan al mismo archivo.

Grafo acíclico

Enlace simbólico o suave: Un nuevo archivo cuyo contenido es el nombre del archivo enlazado.

Archivos

- La programación en UNIX se basa en el uso de un conjunto de funciones que se denominan **llamadas al sistema**. Éstas constituyen un conjunto básico de elementos que permiten construir programas que pueden explotar los recursos del sistema. Mediante llamadas al sistema el programador puede:

- * Utilizar el sistema de archivos.
- * Ejecutar varios procesos de forma concurrente.
- * Comunicar y sincronizar procesos tanto de forma local como en red.

Archivos

- Las llamadas al sistema tienen un formato estándar, tanto en la documentación que suministra el
- manual del sistema como en la forma de invocación. Por ejemplo, la llamada al sistema *read()* tiene el
- siguiente formato:

Archivos

- *int **read**(df, buf, contador)*
- *int df ;*
- *char *buf ;*
- *int contador ;*

- La forma de uso de una llamada al sistema es idéntica a cualquier otra función del lenguaje C. La mayoría de las llamadas devuelven un valor, que suele indicar si la operación se ha efectuado con éxito o no.

Archivos

- Cada proceso en UNIX dispone de 20 descriptores de archivo, numerados de 0 a 19. Por convención, los
- tres primeros descriptores se abren automáticamente cuando el proceso empieza su ejecución, y tienen un
- significado especial:

Archivos

- * descriptor 0: entrada estándar
- * descriptor 1: salida estándar
- * descriptor 2: error estándar

Archivos

- Cada descriptor de archivo posee un conjunto de propiedades privadas y que no depende del tipo de
- archivo que esté asociado al descriptor:

Archivos

- * Desplazamiento (*offset*) del archivo.
- * Un indicador (*flag*) que indica si el descriptor se debe de cerrar de forma automática si el proceso utiliza la llamada al sistema *exec()*.
- * Un indicador que indica si la salida hacia el archivo debe de añadirse al final del mismo.

Descriptor de Archivos

- El *descriptor de archivos o bloque de control de archivos* es un bloque de control que contiene información que el sistema necesita para administrar un archivo .

Descriptor de Archivos

- Puede incluir la siguiente información:
- -Nombre simbólico del archivo.
- -Localización del archivo en el almacenamiento secundario.
- -Organización del archivo (método de organización y acceso).
- -Tipo de dispositivo.

Descriptor de Archivos

- -Datos de control de acceso.
- -Tipo (archivo de datos, programa objeto, programa fuente, etc.).
- -Disposición (permanente contra temporal).
- -Fecha y tiempo de creación.
- -Fecha de destrucción.
- -Fecha de la última modificación.
- -Suma de las actividades de acceso (número de lecturas, por ejemplo).

Descriptor de Archivos

- Los descriptores de archivos suelen mantenerse en el almacenamiento secundario; se pasan al almacenamiento primario al abrir el archivo.
- El descriptor de archivos es controlado por el *sistema de archivos*; el usuario puede no hacer referencia directa a él

Manejo del sistema de Archivos en POSIX

stat, lstat y fstat

Conocido el inodo de un archivo, es posible acceder a la información contenida en él. Para ello se emplean las llamadas al sistema stat, lstat y fstat, las cuales devuelven la información almacenada en el inodo sobre el estado de un archivo concreto.

Manejo del sistema de Archivos en POSIX

La diferencia entre stat y lstat es que lstat devuelve la liga simbólica y stat el archivo que apunta a la líga simbólica.

La diferencia entre stat y fstat, es que la primera reciba como primer parámetro un puntero al path del archivo, mientras la segunda emplea un archivo ya abierto.

Manejo del sistema de Archivos en POSIX

- La sintaxis de las llamadas es:
- int stat (const char *path, struct stat *buf)
- int lstat (const char *path, struct stat *buf)
- int fstat (int fildes, struct stat *buf)

Manejo del sistema de Archivos en POSIX

- Estas llamadas devuelven la información almacenada en una estructura del tipo **struct stat**, la cual está definida en el archivo de cabecera **<sys/stat.h>** y cuyos tipos incluidos se definen en el archivo **<sys/types.h>**

Manejo del sistema de Archivos en POSIX

- Entre sus campos, el que indica el *modo* del archivo, contiene información heterogénea distribuida en sus bits. Para acceder a esa información existen un conjunto de constantes definidas en **<sys/mode.h>** que aplicadas mediante una operación *and* (&), sobre el campo correspondiente, devuelven la información requerida.

Manejo del sistema de Archivos en POSIX

La estructura **stat**, devuelta por las llamadas indicadas, consta de distintos campos, los cuales dependen de la versión de UNIX/Linux que estemos utilizando.

Manejo del sistema de Archivos en POSIX

dev_t st_dev	Número del dispositivo que contiene al *inode*. *Major number* (8 bits de mayor peso) y *minor number* (8 bits de menor peso)
ino_t st_ino	Número de *inode*
mode_t st_mode	16 bits que contienen el modo del fichero
nlink_t st_nlink	Número de enlaces del fichero
uid_t st_uid	Identificador de usuario (UID) propietario del fichero
gid_t st_gid	Identificador del grupo (GID) al que pertenece el propietario del fichero
dev_t st_rdev	Identificador de dispositivo. Sólo tiene sentido para ficheros de dispositivo tipo carácter o bloque
off_t st_size	Tamaño del fichero en bytes
unsigned long st_bksize	Tamaño del bloque en el sistema de ficheros
unsigned long st_blocks	Número de bloques asignados
time_t st_atime	Fecha del último acceso
time_t st_mtime	Fecha de la última modificación del fichero
time_t st_ctime	Fecha del último cambio de la información administrativa. Las fechas están dadas en segundos, con respecto a las 00:00:00 GMT del 1 de enero de 1970

Manejo del sistema de Archivos en POSIX

Con respecto al campo **st_mode**, señalar que los 9 bits de menor peso son los que indican los derechos de acceso para el propietario, el grupo y el resto de usuarios, y los 4 de mayor peso determinan el tipo de archivo acorde con la siguiente tabla:

Manejo del sistema de Archivos en POSIX

1000	Ordinario
0100	Directorio
0010	Especial modo carácter
1100	Especial modo bloque
0001	FIFO
1010	Enlace simbólico

Manejo del sistema de Archivos en POSIX

- Los bits de modo de un archivo, además de los nueve bits de protección, constituyen una máscara de 16 bits.
- - El bit 9(*stiky bit*) indica al kernel que el archivo es un programa con capacidad para que varios procesos compartan su segmento de código.

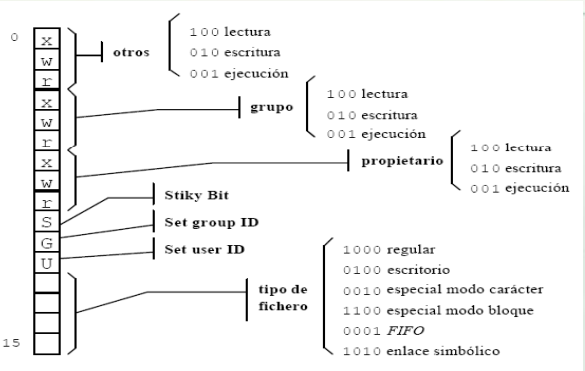
Manejo del sistema de Archivos en POSIX

- - El bit 10(*Set Group ID*) tiene un significado similar al bit 11 pero referido a grupos de usuarios.
- - El bit 11(*Set User ID*) indica al kernel que cuando un proceso acceda a este archivo cambie el UID del proceso por el del propietario del archivo.

Manejo del sistema de Archivos en POSIX

- Se habla en este caso de usuario efectivo
- (*efective UID*) para distinguirlo del usuario real (*real UID*). El identificador de usuario efectivo se emplea siempre para determinar permisos.
- El identificador de usuario real refleja la identidad del usuario. Este mecanismo permite acceder a archivos de otro usuario sobre los cuales no se tiene permiso.

Manejo del sistema de Archivos en POSIX



Manejo del sistema de Archivos en POSIX

- - Los bits 12-15 indican el tipo del archivo. En el archivo de cabecera *<sys/stat.h>* existen un conjunto de macros predefinidas que, dado el modo de un archivo, devuelven el valor verdadero si el archivo es del tipo preguntado :

Manejo del sistema de Archivos en POSIX

- Se definen las siguientes macros POSIX para comprobar el tipo de archivo:
  - S_ISLNK(m)
    - es un enlace simbólico?
  - S_ISREG(m)
    - un archivo regular?
  - S_ISDIR(m)
    - un directorio?
  - S_ISCHR(m)
    - un dispositivo de caracteres?
  - S_ISBLK(m)
    - un dispositivo de bloques?
  - S_ISFIFO(m)
    - una tubería nombrada (fifo)?
  - S_ISSOCK(m)
    - un enchufe (socket)?

Manejo del sistema de Archivos en POSIX

- Existen dos maneras de abrir un archivo, open() y creat(). Antiguamente open() sólo podía abrir archivos que ya estaban creados por lo que era necesario hacer una llamada a creat() para llamar a open() posteriormente. A día de hoy open() es capaz de crear archivos, ya que se ha añadido un nuevo parámetro en su prototipo:

Manejo del sistema de Archivos en POSIX

- int creat(const char *pathname, mode_t mode)
- int open(const char *pathname, int flags)
- int open(const char *pathname, int flags, mode_t mode)

Manejo del sistema de Archivos en POSIX

- La nueva open() es una suma de las funcionalidades de la open() original y de creat().
- Para emplear estas syscalls se suelen incluir los archivos de cabecera:

Manejo del sistema de Archivos en POSIX

#include <sys/types.h>
- #include <sys/stat.h>
- #include <fcntl.h>

- El funcionamiento de open() es el siguiente: al ser llamada intenta abrir el archivo indicado en la cadena “pathname” con el acceso que indica el parámetro “flags”.

Manejo del sistema de Archivos en POSIX

Estos “flags” indican si queremos abrir el archivo para lectura, para escritura, etc. La siguiente tabla especifica los valores que puede tomar este parámetro:

Manejo del sistema de Archivos en POSIX

Indicador	Valor	Descripción
O_RDONLY	0000	El fichero se abre sólo para lectura.
O_WRONLY	0001	El fichero se abre sólo para escritura.
O_RDWR	0002	El fichero se abre para lectura y escritura.
O_RANDOM	0010	El fichero se abre para ser accedido de forma aleatoria (típico de discos).
O_SEQUENTIAL	0020	El fichero se abre para ser accedido de forma secuencial (típico de cintas).
O_TEMPORARY	0040	El fichero es de carácter temporal.
O_CREAT	0100	El fichero deberá ser creado si no existía previamente.
O_EXCL	0200	Provoca que la llamada a open falle si se especifica la opción O_CREAT y el fichero ya existía.
O_NOCTTY	0400	Si el fichero es un dispositivo de terminal (TTY), no se convertirá en la terminal de control de proceso (CTTY).
O_TRUNC	1000	Fija el tamaño del fichero a cero bytes.
O_APPEND	2000	El apuntador de escritura se situa al final del fichero, se escribirán al final los nuevos datos.

Manejo del sistema de Archivos en POSIX

O_NONBLOCK	4000	La apertura del fichero será no bloqueante. Es equivalente a O_NDELAY.
O_SYNC	10000	Fuerza a que todas las escrituras en el fichero se terminen antes de que se retorne de la llamada al sistema. Es equivalente a O_FSYNC.
O_ASYNC	20000	Las escrituras en el fichero pueden realizarse de manera asíncrona.
O_DIRECT	40000	El acceso a disco se producirá de forma directa.
O_LARGEFILE	100000	Utilizado sólo para ficheros extremadamente grandes.
O_DIRECTORY	200000	El fichero debe ser un directorio.
O_NOFOLLOW	400000	Fuerza a no seguir los enlaces simbólicos. Útil en entornos críticos en cuanto a seguridad.

Manejo del sistema de Archivos en POSIX

La lista es bastante extensa y los valores están pensados para que sea posible concatenar o sumar varios de ellos, es decir, hacer una OR lógica entre los diferentes valores, consiguiendo el efecto que deseamos.

Manejo del sistema de Archivos en POSIX

Así pues, podemos ver que en realidad una llamada a creat() tiene su equivalente en open(), de esta forma:

```
open( pathname, O_CREAT | O_TRUNC |
  O_WRONLY, mode )
```

Manejo del sistema de Archivos en POSIX

El argumento “mode” se encarga de definir los permisos dentro del Sistema de Archivos.

La lista completa de sus posibles valores es esta:

Manejo del sistema de Archivos en POSIX

Indicador	Valor	Descripción
S_IROTH	0000	Activar el bit de lectura para todo los usuarios.
S_IWOTH	0001	Activar el bit de escritura para todo los usuarios.
S_IXOTH	0002	Activar el bit de ejecución para todo los usuarios.
S_IRGRP	0010	Activar el bit de lectura para todo los usuarios pertenecientes al grupo.
S_IRGRP	0020	Activar el bit de escritura para todo los usuarios pertenecientes al grupo.
S_IRGRP	0040	Activar el bit de ejecución para todo los usuarios pertenecientes al grupo.
S_IRUSR	0100	Activar el bit de lectura para el propietario.
S_IWUSR	0200	Activar el bit de escritura para el propietario.

Manejo del sistema de Archivos en POSIX

S_IXUSR	0400	Activar el bit de ejecución para el propietario.
S_ISVTX	1000	Activa el "sticky bit" en el fichero.
S_ISGID	2000	Activa el bit de SUID en el fichero.
S_ISUID	4000	Activa el bit de SGID en el fichero.
S_IRWXU	S_IRUSR + S_IWUSR + S_IXUSR	Activar el bit de lectura, escritura y ejecución para el propietario.
S_IRWXG	S_IRGRP + S_IWGRP + S_IXGRP	Activar el bit de lectura, escritura y ejecución para todo los usuarios pertenecientes al grupo.
S_IRWXO	S_IROTH + S_IWOTH + S_IXOTH	Activar el bit de lectura, escritura y ejecución para todo los usuarios.

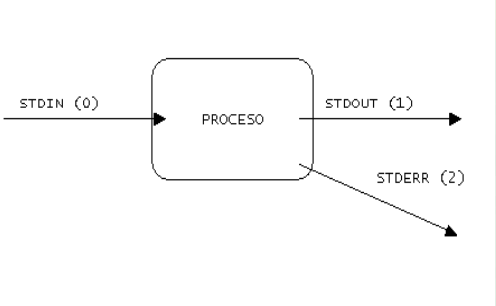
Manejo del sistema de Archivos en POSIX

- Todos estos valores se definen en un archivo de cabecera , por lo que conviene incluirlo:
- #include <sys/stat.h>
- Una llamada correcta a open() devuelve un entero que corresponde al descriptor de archivo para manejar el archivo abierto.

Manejo del sistema de Archivos en POSIX

- Cada proceso maneja una tabla de descriptores de archivo que le permiten manejar dichos archivos de forma sencilla. Inicialmente las entradas 0, 1 y 2 de esa tabla están ocupadas por los archivos STDIN, STDOUT y STDERR respectivamente, es decir, la entrada estándar, la salida estándar y la salida de error estándar:

Manejo del sistema de Archivos en POSIX



Manejo del sistema de Archivos en POSIX

- Cada proceso maneja una tabla de descriptores de archivo que le permiten manejar dichos archivos de forma sencilla. Inicialmente las entradas 0, 1 y 2 de esa tabla están ocupadas por los archivos STDIN, STDOUT y STDERR respectivamente, es decir, la entrada estándar, la salida estándar y la salida de error estándar:

Manejo del sistema de Archivos en POSIX

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main( int argc, char *argv[] )
{
  int fd;

  if( (fd = open( argv[1], O_RDWR )) == -1 )
  {
    perror( "open" );
    exit( -1 );
  }

  printf( "El fichero abierto tiene el descriptor %d.\n", fd );

  close( fd );

  return 0;
}
```

Manejo del sistema de Archivos en POSIX

- El siguiente paso es leer y escribir en los archivos abiertos, para ello se utilizan las llamadas al sistema:
- read()
- write()

- ssize_t read(int fd, void *buf, size_t count)
- ssize_t write(int fd, void *buf, size_t count)

Manejo del sistema de Archivos en POSIX

- La primera de ellas intenta leer “count” bytes del descriptor de archivo definido en “fd”, para guardarlos en el buffer “buf”.
- Al terminar, read() devuelve el número de bytes leídos, por lo que comparando este valor con la variable “count” podemos saber si ha conseguido leer tantos bytes como pedíamos o no.

Manejo del sistema de Archivos en POSIX

- El uso de la función write() es muy similar, basta con llenar el buffer “buf” con lo que queramos escribir, definir su tamaño en “count” y especificar el archivo en el que escribiremos con su descriptor de archivo en “fd”.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512

int main( int argc, char *argv[] )
{
        int fd, readbytes;
        char buffer[SIZE];

        if( (fd = open( argv[1], O_RDWR )) == -1 )
        {
                perror( "open" );
                exit( -1 );
        }

        while( (readbytes = read( fd, buffer, SIZE )) != 0 )
        {
/*              write( STDOUT, buffer, SIZE ); */
                write( STDOUT, buffer, readbytes );
        }

        close( fd );

        return 0;
}
```

Manejo del sistema de Archivos en POSIX

Lectura de un archivo

En ocasiones no queremos posicionarnos al principio de un archivo para leer o escribir, nos interesa posicionarnos en un desplazamiento concreto relativo al comienzo del archivo, o al final de este.

Manejo del sistema de Archivos en POSIX

off_t lseek(int fildes, off_t offset, int whence);

Los parámetros que recibe son “fildes” es el descriptor de archivo, “offset” es el desplazamiento en el que queremos posicionarnos, relativo a lo que indique “whence”, que puede tomar los siguientes valores:

Manejo del sistema de Archivos en POSIX

Indicador	Valor	Descripción
SEEK_SET	0	Posiciona el puntero a “offset” bytes desde el comienzo del archivo.
SEEK_CUR	1	Posiciona el puntero a “offset” bytes desde la posición actual del puntero.
SEEK_END	2	Posiciona el puntero a “offset” bytes desde el final del archivo.

Manejo del sistema de Archivos en POSIX

Ejemplo: Leer un archivo y saltarnos una cabecera de 200 bytes:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512

int main( int argc, char *argv[] )
{
    int fd, readbytes;
    char buffer[SIZE];

    if( (fd = open( argv[1], O_RDWR )) == -1 )
    {
        perror( "open" );
        exit( -1 );
    }

    lseek( fd,200, SEEK_SET );

    while( (readbytes = read( fd, buffer, SIZE )) != 0 )
    {
        write( STDOUT, buffer, SIZE );
    }

    close(fd);

    return 0;
}
```

Manejo del sistema de Archivos en POSIX

Directorios

Lo primero es conocer en que directorio estamos ubicados, es decir el directorio actual de trabajo, la función que permite esto es getcwd(), getcurrent_dir_name() y getwd(), y tienen los siguientes prototipos:

Manejo del sistema de Archivos en POSIX

char *getcwd(char *buf, size_t size); char *get_current_dir_name(void); char *getwd(char *buf);

La función getcwd() devuelve una cadena de caracteres con la ruta completa del directorio de trabajo actual, que almacenará en el buffer “buf”, de tamaño “size”. Si el directorio no cabe en el buffer, retornará NULL, por lo que es conveniente usar alguna de las otras dos funciones

Manejo del sistema de Archivos en POSIX

```
#include <unistd.h>
int main()
{
char buffer[512];
printf( "El directorio actual es: %s\n",
  getcwd( buffer, -1 ) ); return 0;
}
```

Manejo del sistema de Archivos en POSIX

Para movernos a otro directorio:

```
int chdir(const char *path);
int fchdir(int fd);
```

Para crear y borrar directorios tenemos una serie de funciones a nuestra disposición, con prototipos muy familiares:

Manejo del sistema de Archivos en POSIX

```
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
```

rmdir() borra el directorio especificado en “pathname” y exige que éste esté vacío, mkdir() crea el directorio especificado en “pathname”, con el modo de acceso especificado en el parámetro “mode” (típicamente un valor octal como “0755”, etc.).

Manejo del sistema de Archivos en POSIX

```
#include <unistd.h>

int main( int argc, char *argv[] )
{
  char buffer[512];

  printf( "El directorio actual es: %s\n",
          getcwd( buffer, -1 ) );
  chdir( ".." );
  mkdir( "./directorio1", 0755 );
  mkdir( "./directorio2", 0755 );
  rmdir( "./directorio1" );

  return 0;
}
```

Manejo del sistema de Archivos en POSIX

Ahora manipularemos a los directorios mediante las estructuras propias para su manipulación.

Estas son:

DIR *opendir(const char *name);
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);

Manejo del sistema de Archivos en POSIX

Con la primera de ellas conseguimos una variable de tipo DIR en función de una ruta definida por la cadena de caracteres “name”. Una vez obtenida dicha variable de tipo DIR, se la pasamos como parámetro a la función readdir(), que nos proporcionará un puntero a una estructura de tipo dirent, es decir, a la entrada del directorio en concreto a la que hemos accedido.

Manejo del sistema de Archivos en POSIX

En esa estructura dirent tendremos todos los datos de la entrada de directorio a la que estamos accediendo: inodo, distancia respecto del comienzo de directorio, tamaño de la entrada y nombre

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main( int argc, char *argv[] )
{
  DIR *dir;
  struct dirent *mi_dirent;

  if( argc != 2 )
  {
    printf( "%s: %s directorio\n", argv[0], argv[0] );
    exit( -1 );
  }

  if( (dir = opendir( argv[1] )) == NULL )
  {
    perror( "opendir" );
    exit( -1 );
  }

  while( (mi_dirent = readdir( dir )) != NULL )
    printf( "%s\n", mi_dirent->d_name );

  closedir( dir );

  return 0;
}
```

Manejo del sistema de Archivos en POSIX

Dentro de la estructura dirent tenemos un campo en especial que nos será útil: d_name el cual nos proporciona el nombre de los archivos registrados en el directorio.

Estructura del Sistema de Archivos

El sistema de archivos permite organizar la información dentro de los dispositivos de almacenamiento secundario en un formato inteligible para el sistema operativo. Habitualmente, cuando se instala el sistema operativo, los dispositivos de almacenamiento están vacíos.

Estructura del Sistema de Archivos

Por ello previamente a la instalación del sistema de archivos es necesario dividir física o lógicamente los discos en particiones o volúmenes.

Una partición es una porción de un disco a la que se le dota de una identidad propia y puede ser manipulada por el sistema operativo como una entidad lógica independiente

Estructura del Sistema de Archivos

Una vez creadas las particiones el sistema operativo debe crear las estructuras de los sistemas de archivos dentro de esas particiones, para ellos algunos sistemas utilizan las funciones format o mkfs.

Estructura del Sistema de Archivos

El tamaño del sistema de archivos se define en bloques.

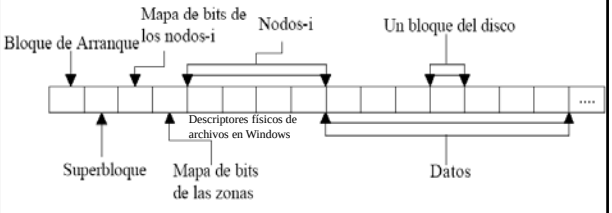
Un bloque se define como una agrupación lógica de sectores de disco y es la unidad de transferencia mínima que usa el sistema de archivos

Estructura del Sistema de Archivos

Cuando se crea un sistema de archivos en una partición de un disco, se crea una entidad lógica autocontenida con espacio para la información de carga del sistema operativo, descripción de su estructura, descriptores de archivo, información del estado de la ocupación de los bloques del sistema de archivos y bloques de datos.

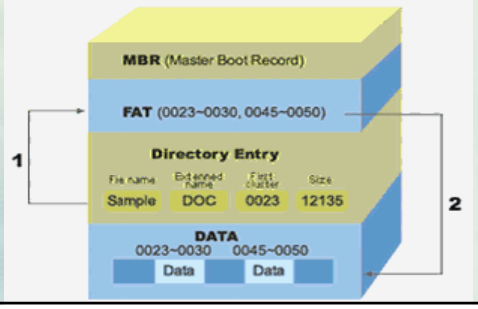
Estructura del Sistema de Archivos

- Estructura de sistema de archivos para UNIX y Windows



Estructura del Sistema de Archivos

- Para MS-DOS



Estructura del Sistema de Archivos

En UNIX cada sistema de archivos tiene un bloque de Carga (BOOT) que contiene el código que ejecuta el programa de arranque del programa almacenado en la ROM.

Estructura del Sistema de Archivos

A continuación del bloque de carga está la metainformación. Esta describe el sistema de archivos y la distribución de sus componentes. El primer componente de la metainformación es el Superbloque el cual contiene toda la información que describe la estructura del sistema de archivos

Estructura del Sistema de Archivos

La información del superbloque indica al sistema operativo las características del sistema de archivos donde están los distintos elementos del mismo y cuanto ocupan.

Estructura del Sistema de Archivos

El Mapa de bits incluye información de la gestión de espacio en el disco, la cual permite al servidor de archivos implementar distintas políticas de asignación de espacio y para reutilizar los espacios liberados.

Estructura del Sistema de Archivos

A continuación se encuentran los descriptores físicos de los archivos, en el caso de UNIX i-nodos o registros en Windows. Cuando se crea un sistema de archivos el sistema operativo habilita un número de descriptores de archivo proporcional al tamaño del dispositivo.

Estructura del Sistema de Archivos

En LINUX por ejemplo se crea un i-nodo por cada 2 bloques de datos.

El último componente del sistema de archivos son los bloques de datos, estos son asignados a los archivos por el servidor de archivos, que establece una correspondencia entre el bloque y el archivo a través del descriptor de archivo.

Mecanismos de asignación

Una de las cuestiones más importantes del diseño y la implementación de un servidor de archivos es cómo asignar los bloques de disco a un archivo y cómo hacerlos corresponder con la imagen del archivo que tiene la aplicación.

Mecanismos de asignación

Este problema se resuelve con lo que tradicionalmente se conoce como mecanismo de asignación, hay varias políticas para esta tarea, existiendo dos claras: **Asignación de bloques contiguos** y **Asignación de bloques no contiguos.**

Mecanismos de asignación

Asignación de bloques contiguos: Con este método, un archivo de 64 KB ocuparía 16 bloques consecutivos en un sistema que use un tamaño de bloque de 4 KB. Es sencillo de implementar y el rendimiento de la E/S es muy bueno, pero si no se conoce el tamaño total del archivo cuando se crea, puede ser necesario buscar un nuevo hueco de bloques consecutivos cada vez que el archivo crece.

Mecanismos de asignación

Además la necesidad de buscar huecos contiguos origina una gran fragmentación externa en el disco, ya que hay muchos huecos no utilizables debido a la política de asignación. Sería necesario entonces compactar el disco muy frecuentemente.

Mecanismos de asignación

Debido a estas desventajas, ningún sistema operativo moderno usa este método, a pesar de que la representación interna del archivo es muy sencilla, basta con conocer el primer bloque del archivo y su longitud.

Mecanismos de asignación

Asignación de bloques no contiguos: Con este método se asigna al archivo el primer bloque que se encuentra libre. De esta forma se elimina el problema de la fragmentación externa del disco y el de la búsqueda de huecos. Además los archivos pueden crecer mientras exista espacio en el disco.

Mecanismos de asignación

Sin embargo este método complica la implementación de la imagen de archivo que se usa en el servidor de archivos. La razón es que se pasa de un conjunto de bloques contiguos a un conjunto de bloques dispersos, debiendo conocer dónde están dichos bloques y en que orden deben recorrerse.

Mecanismos de asignación

Para resolver el problema de mantener un mapa de bloques discontiguos de un archivo, casi todos los sistemas operativos usan una **lista de bloques** en combinación con un **índice**, para optimizar el acceso.

Mecanismos de asignación

En una **lista enlazada** desde cada bloque de un archivo existe un apuntador al siguiente bloque del mismo. En el descriptor de archivo se indica únicamente el primer bloque del archivo. Este método tiene varias desventajas:

Mecanismos de asignación

- No es adecuado para accesos aleatorios, ya que hay que leer todos los bloques para recorrer la cadena de enlaces hasta el destino.

-Puesto que el apuntador al bloque siguiente ocupa espacio (ejem: 4 bytes) y están incluidos en el archivo, el calculo de la longitud real del archivo es más complicado.

Mecanismos de asignación

- Es muy poco fiable ya que la pérdida de un bloque del archivo supone la pérdida de todos los datos del archivo que van detrás de dicho bloque.

Mecanismos de asignación

Las desventajas de la lista enlazada se pueden eliminar si se quitan los apuntadores de los bloques del archivo y se almacenan en un **índice enlazado** gestionado por el servidor de archivos. Cuando se crea un sistema de archivos, se almacena en una parte especial del mismo una tabla que contiene una entrada por cada bloque de disco

Mecanismos de asignación

y que está indexada por número de bloque. Usando esta tabla cada vez que se crea un archivo se incluye en su descriptor el descriptor de la tabla que apunta al primer bloque del archivo. A medida que se asignan nuevos bloques al archivo se apunta a ellos desde la última entrada de la tabla asociada al archivo.

Mecanismos de asignación

Sin embargo una mejor solución es usar un **índice múltinivel** cuyos bloques son apuntados desde el nodo-i que describe al objeto archivo. Con esta solución cada archivo tiene sus **bloques de índice** que incluyen apuntadores a los bloques de disco del archivo.

Mecanismos de asignación

La ventaja de usar índices es que basta con traer a memoria el bloque de índices donde está el apuntador a los datos para tener acceso a cada bloque de datos.

Mecanismos de asignación

sin embargo la mejor solución es la combinación de un **sistema de índices puro** con un **sistema de índices multinivel**, que es el que usa UNIX y LINUX actualmente. Un i-nodo tiene varios apuntadores directos a bloques de datos y tres apuntadores a bloques de índices.

Mecanismos de asignación

Esto permite almacenar archivos pequeños sin necesitar bloques de índices.

Permite accesos aleatorios muy grandes.

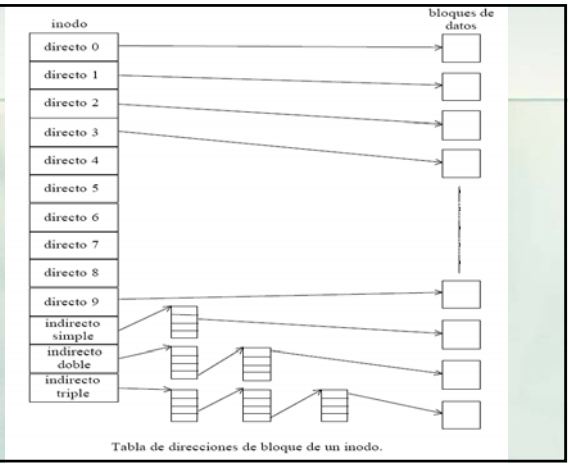


Tabla de direcciones de bloque de un inodo.

Mecanismos de asignación

Pero se tiene una desventaja, hay que acceder al i-nodo para tener las direcciones de los bloques de disco y luego leer los datos, para eso UNIX mantiene los i-nodos de los archivos abiertos en una tabla de memoria.

Mecanismos de asignación

Para evitar este acceso extra Windows almacena datos en el mismo descriptor de archivo, de esta forma archivos pequeños pueden leerse completamente con un único acceso a disco.