

# Sistemas de archivos: Archivos y directorios

Gunnar Wolf



# Índice

- 1 Semántica de archivos
- 2 Tipos de archivo
- 3 Métodos de acceso
- 4 Organización de archivos
- 5 Sistemas de archivos remotos



# Sistemas de archivos

- Gestión del espacio de almacenamiento
- Probablemente el rol con más visibilidad de los que cubren los sistemas operativos
  - Comprendido casi universalmente por los usuarios



# Abstracción primaria

- El modelo primario, con el cual todos estaremos familiarizados, es el del *directorio jerárquico*
- Unidad de almacenamiento visible al usuario: *Archivo*
  - No podemos grabar información como no sea en un archivo
  - Se ubica por un *nombre* (o por una *ruta*) en el *directorio*
- Una de las abstracciones más longevas en la historia de la computación



## Tipo de datos abstracto

- Cuando hablamos de *manejo de archivos*, necesariamente lo hacemos a través de un *tipo de datos abstracto*
  - Esto es, en el lenguaje y bajo el paradigma que sea, un tipo de programación *orientada a objetos*
  - Una estructura de datos (*archivo*) es *opaca* (no se puede acceder a sus contenidos más que a través de operaciones que trabajan *en su interior*)
  - Hay una serie de *operaciones definidas*
- Cada sistema operativo nos presenta un *conjunto de operaciones* que define la *semántica de archivos* que maneja



# Operaciones con archivos

Hay algunas diferencias entre sistemas operativos, pero en general tendremos las siguientes operaciones disponibles:

- Crear
- Borrar
- Abrir
- Cerrar
- Leer
- Escribir
- Reposicionar (*seek*)



## Operaciones con archivos: *Crear*

- Asigna una entrada en el *directorio* para un nuevo archivo
- Posiblemente, también asigna espacio en el dispositivo para sus contenidos.



## Operaciones con archivos: *Borrar*

- Elimina al archivo del directorio
- Si corresponde, libera el espacio del dispositivo que el archivo emplea





## Operaciones con archivos: *Abrir*

- Verificar si tenemos acceso para el *modo de acceso* indicado
- Verificar si el medio soporta el *modo de acceso*
  - Por ejemplo, no podemos abrir para escritura un archivo en un disco de sólo lectura
  - Aunque tengamos permisos
- Asigna un *descriptor de archivos* a la relación entre el proceso y el archivo en cuestión
- Es necesario abrir un archivo para todas las operaciones que realizaremos con sus datos



## Operaciones con archivos: *Cerrar*

- Indica al sistema que el proceso terminó de usar al archivo
- El sistema vacía los buffers a disco
- El sistema elimina la relación archivo-proceso de las *tablas* activas
- Se invalida al *descriptor de archivo*
- Si un proceso cierra un archivo y quiere emplearlo *de cualquier manera*, tiene que volverlo a abrir explícitamente.



## Operaciones con archivos: *Leer*

- Indicamos al sistema un buffer donde poner el resultado
- El sistema copia el siguiente *pedazo* de información en el archivo hacia el buffer
  - ¿Qué es *siguiente*?



## Operaciones con archivos: *Leer*

- Indicamos al sistema un buffer donde poner el resultado
- El sistema copia el siguiente *pedazo* de información en el archivo hacia el buffer
  - ¿Qué es *siguiente*? Lo que indique el *apuntador de última posición*
  - ¿Qué es *pedazo*?



## Operaciones con archivos: *Leer*

- Indicamos al sistema un buffer donde poner el resultado
- El sistema copia el siguiente *pedazo* de información en el archivo hacia el buffer
  - ¿Qué es *siguiente*? Lo que indique el *apuntador de última posición*
  - ¿Qué es *pedazo*? Un bloque de longitud fija, una línea de texto — dependiendo del modo en que esté abierto el archivo / solicitada la lectura



## Operaciones con archivos: *Escribir*

- Indicamos un buffer al sistema
- El sistema copia de dicho buffer al archivo
- ¿A qué parte del archivo?

**Truncar** Descartar al contenido actual del archivo (pero no su entrada en el directorio) y reemplazar por lo que indique el buffer

**Agregar** (*append*) Se guarda la información al final del archivo ya existente

**Escribir tras la última posición** Se puede seguir un apuntador análogo al que presentamos en la lectura



## Operaciones con archivos: *Reposicionar (seek)*

- Lectura y escritura se hacen siguiendo a un apuntador
- El apuntador puede ser *reposicionado* arbitrariamente dentro del archivo
  - O, si está abierto en modo de escritura, a veces fuera del mismo
  - No todos los sistemas lo soportan, pero puede emplearse para crear un archivo muy grande empleando *archivos dispersos*:
    - Se crea un archivo vacío
    - Reposicionamos a cierta posición lejana (digamos, +50MB)
    - Grabamos un sólo bloque



## Semántica de unidad de cinta

- La forma en que opera este conjunto de funciones presenta una semántica en que cada archivo se comporta como si fuera una unidad de cinta
  - Apuntador → Cabeza lectora
  - Posición actual, mas rebobinado/adelantado
  - Lectura/escritura secuencial





# Tablas de archivos abiertos

- Cuando se abre un archivo, se crean entradas para representarlo en dos diferentes tablas

**Global** Todos los archivos abiertos en el sistema

- La referencia debe seguirse manejando por un descriptor global: Probablemente, por *i-nodo* (lo veremos a detalle posteriormente)

**Por proceso** Los archivos empleados por cada uno de los procesos

- La referencia a cada archivo se hace por *descriptor de archivo*



## Tablas de archivos abiertos

- *Semántica* (¿cuál es el comportamiento?) al haber múltiples procesos abriendo un archivo:
  - Depende del sistema operativo
- Estas tablas no deben caer en *redundancia*
  - Cada una tiene información distinta y específica



# Tabla global de archivos abiertos (1)

**Conteo de usuarios del archivo** El sistema debe saber cuántos procesos *dependen* del archivo y de qué manera. Se emplea, por ejemplo, para determinar si una unidad puede ser *desmontada*, para saber si un archivo puede ser abierto para escritura, etc.

**Modos de acceso** Aunque un usuario tenga los *permisos* necesarios para determinado acceso al archivo, el sistema puede negarlo si esto lleva a *inconsistencias* (p.ej. dos procesos abriendo un mismo archivo para escritura)



## Tabla global de archivos abiertos (2)

**Ubicación en disco** El o los bloques físicos en disco donde se ubica cada fragmento del archivo, evitando que cada proceso tenga que consultar el directorio (y estructuras relacionadas) para cada acceso

**Información de bloqueo** Si el modo de acceso requiere sincronización explícita (manejo de *bloqueo*), puede representarse en la tabla global.



## Tabla por proceso de archivos abiertos

- Descriptor de archivo** Relación entre el archivo abierto (típicamente especificado por *nombre*) y un identificador numérico con que lo manejamos *dentro del proceso*.  
Un mismo archivo tendrá diferente *descriptor de archivo* en cada proceso.
- Permisos** Modos válidos de acceso para un archivo. *No es igual* a los permisos del archivo *en disco* — Es la intersección de dichos permisos con el *modo* en el que fue abierto el archivo.



# Bloqueos

- Los archivos pueden emplearse como mecanismo de comunicación entre procesos
- Incluso a lo largo del tiempo
- Pero también pueden causar problemas de sincronización
  - Pueden ser abiertos por procesos no relacionados
  - Al ser manipulados de forma concurrente, pueden llevar a corrupción o pérdida de datos
- Podemos manejar diferentes tipos de bloqueo (o *candado*) sobre los archivos
  - En algunos sistemas, incluso sobre *rangos* dentro de cada uno de ellos
  - Compartido vs. exclusivo
  - Mandatorio vs. consultivo



# Compartido vs. exclusivo

## Compartido (*shared lock*)

- Típicamente empleado para asegurar la lectura concurrente
- Varios procesos pueden adquirir el bloqueo compartido a la vez
  - Esto indica, varios procesos están *leyendo* del archivo
- Tienen la *expectativa* de que el archivo no sufrirá modificaciones

## Exclusivo (*exclusive lock*)

- Un sólo proceso puede adquirirlo a la vez
- Indica que el archivo va a ser modificado



## Candados *mandatorio* vs. *consultivo*

### Mandatorio u obligatorio (*mandatory locking*)

- Una vez que un proceso adquiere este candado, el sistema operativo impone las restricciones a todos los demás procesos
- Independientemente de si estos conocen o no de la posibilidad de que este bloqueo se presente





## Candados *mandatorio* vs. *consultivo*

### Asesor o consultivo (*advisory locking*)

- Es manejado exclusivamente entre los procesos involucrados
- Un proceso que no sepa del bloqueo *consultivo* puede brincárselo
- Pero es mucho más *ligero* al sistema operativo



## ¿Recuerdan las *primitivas de sincronización*?

- Los archivos son *recursos* gestionados por el sistema operativo, como los casos que vimos en la unidad de *administración de procesos*
- Compartido / exclusivo: Muy parecida al *patrón lectores/escritores*
- Mandatorio / consultivo: Podríamos manejar a los *monitores* como mandatorios, y a los *mutexes* y *semáforos* como consultivos
- Corolario: ¡Administrando archivos y bloqueos también podemos caer en bloqueos mutuos!



## Matriz incompleta, explicativa

- De estos  $2 \times 2$  candados, esperaríamos que hubiera 4 tipos de candado
- No todos los sistemas implementan todas las posibilidades
- Como regla general, en Windows se maneja *bloqueo obligatorio* y en Unix *bloqueo consultivo*



# Índice

- 1 Semántica de archivos
- 2 Tipos de archivo
- 3 Métodos de acceso
- 4 Organización de archivos
- 5 Sistemas de archivos remotos



## ¿Qué es el *tipo* de un archivo?

- Según el tratamiento que deba dársele para que tenga sentido, un archivo puede ser de distintos *tipos*
  - Un *documento de texto* puede ser abierto por un editor, un *ejecutable* por el *módulo cargador* del sistema operativo, una imagen por un visor de fotografías, etc.
- No debemos intentar abrir un archivo como el tipo equivocado
  - Puede ir llevar al desconcierto del usuario
  - Puede llevar a pérdidas económicas (p.ej. imprimir un archivo binario, o ejecutar un virus *disfrazado* de algo inocuo)



# Mecanismos para identificar el tipo de un archivo

Los principales mecanismos para distinguir el tipo de un archivo son a través de:

- Extensión
- Metadatos externos
- Números mágicos



## Identificación de tipo por *extensión*

- Empleado en los sistemas derivados de CP/M (incluye MS-DOS, VMS, Windows)
- El nombre de un archivo se divide en dos porciones: El *nombre* y la *extensión*
- La extensión identifica al tipo de archivo; cuando el SO lo identifica, lo maneja acorde
- El esquema original identificaba principalmente a los *ejecutables*: .COM, .EXE (y en CP/M, .CMD, o en MS-DOS .BAT), y...
  - Al pasar a una interfaz gráfica, dar *doble click* sobre cualquier archivo causa lanzar al programa que lo sepa manejar
  - Registro de aplicaciones por tipo de archivo



## Identificando por extensión: Seguridad

- Uno de los criterios de diseño de Windows es presentar una interfaz *amable* al usuario, ocultando detalles técnicos
- Por ejemplo, muestra un icono en vez del tipo de archivo
  - Y oculta a la extensión, que resulta *redundante*
  - ... Pero las extensiones forman ya parte del *inconsciente colectivo*
- Un virus/troyano se autoenvía por e-mail a toda la *lista de contactos* de mi amigo, como archivo adjunto
  - Emplea el nombre `inocente.png.exe`
  - El lector de correo, *amigablemente*, esconde la extensión `.exe`
  - Yo veo a un archivo `inocente.png`, y lo abro confiado
  - **¡Bum!** Estoy infectado.





## Identificación de tipo por *metadatos externos*

- El sistema empleado por la Macintosh desde 1984
- Separa a todos los archivos en dos *divisiones* (*forks*):
  - División de datos (*Data fork*) Los datos que *propiamente* constituyen al archivo
  - División de recursos (*Resource fork*) Información *acerca del archivo* de utilidad para el sistema
- Incluye datos fundamentales para el entorno gráfico *amigable*
  - Icono
  - Posición de la ventana al ser abierta
  - Cadenas de traducción
  - *Creador* → El *programa* que creó al archivo (y será usado para abrirlo si el usuario le da *doble click*)



## Identificación de tipo por *números mágicos*

- Mecanismo empleado por sistemas Unix
- El sistema mantiene una lista compilada de las *huellas digitales* que le permiten identificar a los archivos que maneja
  - El administrador local puede *ampliar* la lista como lo requiera



## Ejemplos de *números mágicos*

- Los archivos tipo *Formato de Intercambio Gráfico* (GIF) comienzan con la cadena GIF87a o GIF89a (dependiendo de la versión)
- Los archivos de descripción de páginas *PostScript* comienzan por %!, y los del *Formato de Documentos Portátiles* (PDF) por %PDF
- Un documento XML inicia frecuentemente con <!DOCTYPE
- Pueden no estar *anclados* al principio, sino en un punto específico del primer bloque



# Números mágicos: El *hashbang*

- La identificación por números mágicos incluye al mecanismo *hashbang* (#!)
- Cuando un archivo inicia por #!, el sistema sabe que debe ejecutar *el comando indicado* en la primer línea, y *alimentarlo con el archivo entero*
  - De ese modo, al ejecutar, por ejemplo, un archivo que inicia por `#!/usr/bin/perl` hace que sea ejecutado por el intérprete del lenguaje Perl



# Índice

- 1 Semántica de archivos
- 2 Tipos de archivo
- 3 Métodos de acceso**
- 4 Organización de archivos
- 5 Sistemas de archivos remotos



# Estructura de los archivos

- Prácticamente todos los archivos que manejemos responden a determinado *formato*
  - Esto es, tienen determinada estructura
  - Los datos pueden *analizarse significativamente* cuando están estructurados
- ¿No puede el sistema operativo asistir ofreciendo formatos de archivo estructurados preestablecidos?
  - Lo hicieron en algún momento
  - Hoy en día, es ya muy raro



# Ventajas de un archivo estructurado por el OS

- El usuario no puede corromper un archivo
  - Sea por error de programación en la aplicación, por acceso no sincronizado, . . .
  - El OS ofrece acceso al archivo por medio de un API
  - Puede ofrecer garantías de atomicidad, estructura, ordenamiento, . . .
- Empleado en muchos sistemas operativos de mainframe en los 1960, 1970
  - IBM CICS (1968), IBM MVS (1974), DEC VMS (1977)



## ¿Y por qué decayó su uso?

- Los formatos pueden resultar demasiado rígidos
  - Dificultad para representar nuevos tipos de datos
  - *No portables*: Su correcta interpretación resulta dependiente del OS en cuestión
    - En algunos casos, de la versión
- Son tareas que pueden ya delegarse a programas dedicados (*gestores de bases de datos*)
  - Ejecutando en *espacio de usuario* — Requieren menos privilegios
  - Pueden responder a solicitudes locales o por red
- O *subsistemas particulares*
  - Como pronto veremos, los *subdirectorios* en nuestros sistemas de archivos son simplemente archivos de tipo especial
  - Gestionados directa y exclusivamente por el SO (e inaccesibles directamente al usuario)





## Remanentes de los archivos estructurados

En CP/M y sus derivados (incl. MS-DOS, Windows), un archivo puede ser abierto en modo *de texto* o en modo *binario*

**Modo de texto** Un cambio de línea es indicado por dos caracteres:

CR+LF (ASCII 13+10)

- Se *espera* que el contenido de cada línea sea ASCII *imprimible* (caracteres 32-127)
  - Actualizado: Mas *extensiones nacionales* (p.ej. caracteres acentuados)
- Dice la leyenda, para evitar demandas de patentes por interoperar con AT&T

**Modo binario** «*Todo vale*»

- *Todo archivo* puede ser abierto en modo binario



# Métodos de acceso

- El SO ya no ofrece / impone estructura a los archivos
- Sin embargo, sí ofrece diferentes *mecanismos de acceso*
  - Acordes a diferentes aplicaciones / formas de uso





# Usos para el acceso secuencial

- El acceso secuencial es la forma *más natural* de manipular archivos
  - *Más natural* en el sentido de que es la semántica ofrecida directamente
- Usos esperables:
  - Lectura/ejecución de binarios, bibliotecas
  - Documentos (p.ej. texto)
  - Estructuras anidadas (p.ej. XML, JSON, YAML)
- Es terriblemente ineficiente para datos *estructurados*
  - No conocemos de antemano el punto de inicio o finalización de cada registro
  - En bases de datos: Un *barrido secuencial* lleva al peor de los rendimientos posibles



## Acceso aleatorio

- Puede programarse directo en la aplicación, o emplear *gestores*
  - Ya sea en biblioteca embebida (SQLite) o como un proceso independiente
- La semántica que nos ofrece el sistema operativo permite *brincar* a puntos arbitrarios del archivo
  - Y, a diferencia de una unidad de cinta *real*, no incurrimos en demora



## Acceso aleatorio: Ejemplo

- El *descriptor de archivo* `FD` apunta a un archivo con 2000 registros de 75 bytes cada uno
- Tenemos la variable `registro`, un buffer de 75 bytes
- Queremos el registro 65
- *Reposicionamos* el apuntador a  $65 \times 75$ :  
`seek (FD, 4875)`
- Leemos los siguientes 75 bytes a nuestra variable:  
`read (FD, *registro, 75)`



## Archivo de acceso aleatorio

	Nombre	Apellido	Teléfono	Correo	UltimaSesion	UsuarioDesde
0	...	...	...	...	...	...
4800	José	Chávez	5154-4553	chavez@aquí.no.es	2013.04.05	2012.01.15
4875	Gonzalo	Oliva				
4950	Raquel	Domínguez		rdomgz@aca.si.es		
5025	...	...	...	...	...	...
150000	...	...	...	...	...	...

Figura: Archivo de acceso aleatorio



## Acceso relativo a índice

- A últimos años se han popularizado los *gestores de base de datos no estructurados, orientados a texto u orientados a documentos*
  - Han adoptado el nombre genérico *NoSQL*
- Pueden guardar registros de *tamaño variable* y sin una estructura interna predefinida
  - Cada registro puede tener conjuntos de datos distintos
- En general, operan con un archivo *corto*, que mantiene únicamente una tabla pequeña
  - Identificador de registro
  - Dirección de inicio
  - Dirección de fin
- Y un archivo *largo*, que contiene los datos propiamente





## Acceso relativo a índice

Apellido	Inicio	Tamaño
Chávez	0	132
Domínguez	163	200
Godoy	428	62
Oliva	132	31
Vázquez	408	20
Zapata	363	45

63  
 83  
 103  
 143  
 163  
 183

... fono##5154-4553;E  
 mail##chavez@aqui.no  
 .es;UltimaSesion##20  
 13.04.05;Nombre##Gon  
 zalo;Apellido##Oliva  
 ;Nombre##Raquel;Apel  
 lido##Domínguez;E...

82  
 102  
 122  
 142  
 162  
 182  
 202

Figura: Acceso relativo a índice: Un índice apuntando al punto justo de un archivo sin estructura



## Acerca del acceso *relativo a índice*

- El índice entero puede típicamente mantenerse en memoria, el tiempo de acceso es muy bajo (igual al de *acceso directo*)
  - Pero con mayor flexibilidad en el *esquema de datos*
- Optimizado para operaciones *mayormente de lectura* (o de agregación al final)
  - Las modificaciones y remociones crean *hoyos* (fragmentación)
- Es necesario *asegurar la sincronía* entre ambos archivos
  - Si se pierde la sincronía, se corrompe la información



## Acceso mapeado a memoria

- Caso especial del acceso aleatorio
- El programa de usuario *no usa* la interfaz *tradicional* de cinta, sino que pide la inclusión de determinado archivo *dentro de su espacio de memoria*

```
from os import path
from mmap import mmap
filename = 'un_archivo' #...
if not path.isfile(filename):
    raise IOError, 'El archivo %s no existe!' % filename
fh = open(filename, 'r+')
datos = mmap(fh.fileno(), 0)
datos[50:74] = 'Escribiendo directamente'
print datos[0:100]
```

Más detalles: man 2 mmap



## Transferencias *orientadas a bloques*

- En los tres modos de acceso, las solicitudes parecen ser del *rango de bytes* solicitados, sean cuantos sean
- En realidad, es una abstracción que nos da el sistema operativo — Las transferencias son siempre hechas en *bloques* de tamaño definido por el hardware
  - Históricamente (y por más de 50 años), 512 bytes
  - Hoy en día (2009–), 4K (*Formato Avanzado*)



# Las transferencias en bloques y el rendimiento

- Lecturas de registros contiguos en el mismo bloque → Servidas de caché
- Lectura aleatoria, o escritura: Deben transferirse bloques enteros
- Escritura a un punto no en *caché*:
  - Una lectura (para obtener el *contexto*, el resto del bloque)
  - Una escritura (del bloque entero modificado)



# Las transferencias en bloques y el rendimiento

- Conviene diseñar estructuras de datos *alineadas al tamaño del bloque*
  - Un registro de 400 bytes en un dispositivo con bloques de 512 bytes *casi siempre* (>75 %) requerirá la transferencia de dos bloques
  - Incluso un registro pequeño (p.ej. 7 bytes) puede caer en la frontera entre bloques (aunque sea sólo 1.3 % de las veces)



# Índice

- 1 Semántica de archivos
- 2 Tipos de archivo
- 3 Métodos de acceso
- 4 Organización de archivos**
- 5 Sistemas de archivos remotos



## Pero hablamos de *muchos* archivos. . .

- Hasta ahora hemos visto cómo se trabaja dentro de *un* archivo
- Sin embargo, esta unidad se refiere a *sistemas* de archivos
- Tenemos que considerar cómo se organizan *numerosos* *archivos* dentro de *un mismo medio*





## Pero hablamos de *muchos* archivos. . .

- Hasta ahora hemos visto cómo se trabaja dentro de *un* archivo
- Sin embargo, esta unidad se refiere a *sistemas* de archivos
- Tenemos que considerar cómo se organizan *numerosos* *archivos* dentro de *un mismo medio*
- Hoy en día, eso significa hablar de *directorios*
  - Aunque existen otras formas de organización





# Sistema de archivos plano

- Primer acercamiento: Permitir que varios archivos existan en el mismo disco, *en el mismo espacio organizacional*
- Sin *jerarquía de directorios*
- Requisito de unicidad global de *nombres*
- Uso de discos (en mainframes): A corto / mediano plazo (no para almacenamiento *permanente*)



## Primeros sistemas en computadoras personales

- Computadoras personales: Sistemas de archivos basados en *discos flexibles*
  - En un principio, entre 80 y 160KB
- Si un usuario requería mantener una división temática, podía separar su trabajo en *discos flexibles* distintos
- IBM PC: El concepto de directorios aparece con la IBM XT (1983)
  - Al llegar el soporte para discos duros (10MB)



## Distintas abstracciones

- El sistema MFS, en la primer versión de la Apple Macintosh, presentaba la *ilusión* de directorios
  - Comparables a *etiquetas* (pero únicamente uno por archivo)
  - Mantenía el requisito de unicidad de nombre de archivo por disco
  - Algunos componentes del sistema *modelaban* los directorios, otros (p.ej. los diálogos) no



# Sistemas de archivos planos hoy

- No han desaparecido, ni son sólo para bajos volúmenes de información
- Por ejemplo, el sistema de *almacenamiento en la nube* Amazon S3 (Simple Storage Service)
  - Maneja únicamente *objetos y cubetas* (similar a *archivos y unidades*)
  - Permite referirse a un objeto o conjunto de objetos empleando *filtros* sobre una cubeta



## Directorios de *profundidad fija*

- Primeras implementaciones: Separación funcional únicamente
- Un sólo nivel
- Nombres de (lo que ahora conocemos como) directorios:
  - **MFD** *Master File Directory* (directorio raíz)
  - **UFD** *User File Directory* (directorios de usuarios del sistema)
- Resuelve el problema del *nombre global único*
- Permite almacenar *mejor* los proyectos a largo plazo
  - Pero sigue siendo relativamente limitado (¿ante las costumbres actuales?)
- Dificulta la colaboración entre usuarios
  - ¿Cómo pueden dos usuarios trabajar en un mismo proyecto?



## Directorios de *profundidad fija*

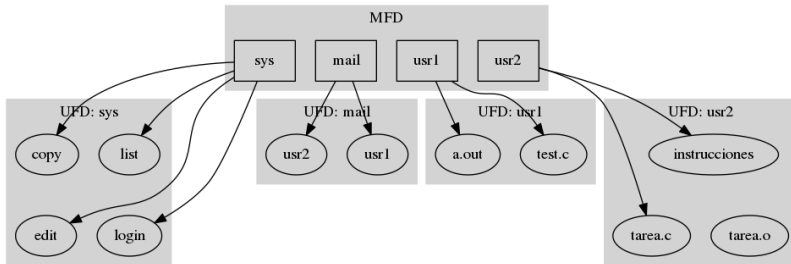


Figura: Directorio simple, limitado a un sólo nivel de profundidad





## Directorios estructurados en *árbol*

- Podemos verlo como paso natural obvio
- Puede permitir múltiples niveles jerárquicos, o incluso una *jerarquía ilimitada*
  - Algunos OSs limitan a cierto número (alto) de niveles *preventivamente*, presentando una *ilusión* de ser ilimitados



# Directorios estructurados en árbol

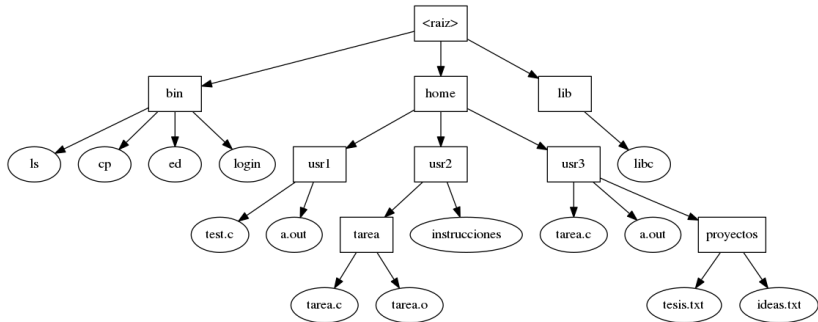


Figura: Directorio estructurado en árbol



## Consecuencias del directorio como un árbol

- Cada usuario, y el sistema entero, estructura su información siguiendo *criterios lógicos propios*
  - Almacenamiento definitivamente visto para el largo plazo
- Nacen las *rutras de búsqueda*
  - Programas y bibliotecas del sistema pueden estar en diferentes lugares
  - El usuario puede tener programas propios
  - El sistema emplea una *ruta de búsqueda* para encontrarlos *por su nombre*
  - Ejemplo Unix:  
`/usr/local/bin:/usr/bin:/bin:~/bin`
  - Ejemplo Windows:  
`c:\WINDOWS\system32;c:\WINDOWS;`  
`c:\WINDOWS\System32\Wbem`

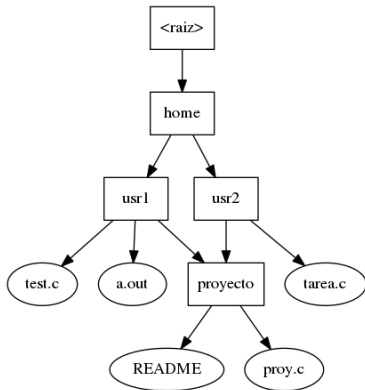


## El directorio como un *grafo dirigido*

- Muchas veces vemos al directorio como un árbol
- En muchos de nuestros sistemas, estrictamente hablando, es un *grafo dirigido*
  - *Superconjunto* de un árbol
  - Un mismo nodo puede tener varios *directorios padre*



## El directorio como un *grafo dirigido*



**Figura:** Directorio como un *grafo dirigido acíclico*: El directorio proyecto está tanto en el directorio /home/usr1 como en el directorio /home/usr2





## Detalles extra en los grafos dirigidos

- En la gráfica anterior, aparecen dos entradas *curiosas* en cada directorio: `.` y `..`
- Encontramos esta semántica tanto en sistemas tipo Unix como tipo Windows
- Son directorios, pero no apuntan a una estructura nueva
  - Apuntan a directorios *ya existentes*
  - `.` apunta *al mismo directorio que lo contiene*
  - `..` apunta al *directorio padre* de donde está contenido
- ¿Y qué pasa con `proyecto`?
  - `..` sólo puede apuntar a *un* lugar (es una sola entrada)



## Ligas duras y ligas simbólicas

- Un archivo puede estar en *más de un lugar* en el directorio
  - Pero un directorio es una entidad un tanto particular
- Esto se expresa empleando las *ligas duras* y *ligas simbólicas*
  - En Windows: Los *accesos directos* son el *pariente pobre* de las ligas simbólicas





# Las ligas duras

- Una entrada en el directorio es meramente la relación entre una *ruta y nombre* de archivo y un *i-nodo*
  - Por ahora, basta decir que un *i-nodo* es un número único, que *apunta* a los *datos* de cada archivo
- Llamamos *liga dura* a que haya más de una entrada en el directorio apuntando al mismo archivo
  - No hay *indirección*, no se refieren unos a otros — Es el mismo objeto, el mismo archivo
  - Si borramos uno, el otro sigue funcionando
  - Si modificamos uno, el otro también se modifica



## Ligas duras: Restricciones

Las ligas duras presentan dos restricciones principales:

- 1 No pueden hacerse ligas duras *fuera del sistema de archivos* (del disco o partición actual)
  - De otro modo, no podría ser el mismo objeto
- 2 No pueden hacerse ligas duras a directorios
  - Formalmente puede haberlas (de hecho, . y .. lo son)
  - Sólo el administrador puede crearlas
  - Pueden causar problemas (veremos en un momento: *Recorrer los directorios*)



## Ligas simbólicas (sistemas Unix)

- Otro mecanismo para que un objeto *aparezca* en más de un punto del directorio
- Es un archivo *especial* que incluye el nombre de otro objeto (¿archivo?) destino
  - Cuando es abierto, *el SO* sigue la indirección
- Son más flexibles, pero menos robustas, que las ligas duras
  - *Pueden* hacerse ligas simbólicas a objetos en otros sistemas de archivos y a directorios
  - Pero el archivo *referido* es el único *real*
    - Si el archivo referido se elimina o renombra, las ligas simbólicas *se rompen*



## El *pariente pobre*: Accesos directos

- Las *ligas duras* existen tanto en Unix como en Windows
  - Aunque en Windows *casi* nunca son empleadas
- Las ligas simbólicas sólo existen en Unix
- Son una herramienta muy útil y poderosa — En Windows 95 se introdujo el *acceso directo* buscando implementar parte de su funcionalidad
  - Son archivos *estándar*
  - Extensión `.lnk`
  - Pensados principalmente para el escritorio y el menú de inicio
- ¿Principal diferencia? No se *siguen* automáticamente
  - Tiene que hacerlo la aplicación — Y pocas lo hacen
  - Un *acceso directo* puede abrirse como un archivo; leer una *liga simbólica* lee directamente del archivo referido



## Volviendo a nuestro grafo dirigido

- No podemos ya decir *formalmente* que sea un grafo dirigido *acíclico*
  - . y . . crean ciclos
  - Aunque con esa excepción hecha, podría ser visto como uno
- `proyecto` está en realidad *sólo* en `/home/usr2`
  - `/home/usr1/proyecto` es una liga simbólica hacia él
    - Pero es transparente a las aplicaciones
  - *Podría* haber ligas duras, con un archivo en varios directorios, o incluso con distintos nombres en el mismo



## Operaciones con directorios

- Para trabajar con los directorios, tenemos una semántica básica similar a la que manejamos para los archivos
  - Varias de las operaciones son análogas cercanas
- Operaciones básicas: (C)
  - Abrir (`dirstream = opendir(dirname)`) y cerrar (`closedir(dirstream)`)
    - El *flujo de directorio* (*directory stream*) es equivalente al *descriptor de archivo*
    - Podemos *rebobinar* con `rewinddir(dirstream)`
  - Listado de objetos en el directorio: Iterar sobre sus objetos (`dirent = readdir(dirstream)`)
  - Buscar un objeto en particular: Filtrar sobre del listado
  - Crear, eliminar o renombrar un objeto: Si bien son *escrituras al directorio*, se implementan a través de funciones de manejo de archivos



## Ejemplo de operaciones con directorios

```
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
    struct dirent *archivo;
    DIR *dir;
    if (argc != 2) {
        printf("Indique el directorio a mostrar\n");
        return 1;
    }
    dir = opendir(argv[1]);
    while ((archivo = readdir(dir)) != 0) {
        printf("%s\t", archivo->d_name);
    }
    printf("\n");
    closedir(dir);
}
```



## Recorriendo los directorios (*directory traversal*) (1)

- Una operación muy frecuente es el *recorrido* (*traversal*) de directorios
  - Para agrupar una estructura completa en un archivo comprimido
  - Para copiar el contenido completo de un directorio a otro medio
  - ...
- Sistema de archivos plano: Trivial
  - El programa recién visto ilustra cómo





## Recorriendo los directorios (*directory traversal*) (1)

- Una operación muy frecuente es el *recorrido* (*traversal*) de directorios
  - Para agrupar una estructura completa en un archivo comprimido
  - Para copiar el contenido completo de un directorio a otro medio
  - ...
- Sistema de archivos plano: Trivial
  - El programa recién visto ilustra cómo
- Sistemas de profundidad fija o árbol: Sencillo
  - Verificación de tipo de archivo, separación en una función, recursión



## Recorriendo los directorios (*directory traversal*) (2)

- Sistemas de grafos dirigidos: ... Se complica



## Recorriendo los directorios (*directory traversal*) (2)

- Sistemas de grafos dirigidos: ... Se complica
- Queremos mantener una tabla de *archivos visitados* en memoria
  - Para evitar caer en un ciclo sin fin
  - La tabla debe registrarse *por i-nodo*, no por nombre de archivo  
→ Los nombres de archivo no serán iguales
- Pueden ser decenas o cientos de miles de archivos en un uso típico



# Montaje de sistemas de archivos

- Un sistema puede trabajar con *un sólo* sistema de archivos
- Pero muchas veces trabaja con varios
  - Distintos medios físicos → sistemas de archivos independientes
  - Diferentes objetivos → sistemas de archivos especializados para cada uno
  - Abstracciones de sistemas de archivos no físicos
  - Razones administrativas (p.ej. confinar espacio máximo ocupado)
- ¿Cómo presenta el sistema operativo a los diferentes sistemas de archivos presentes?

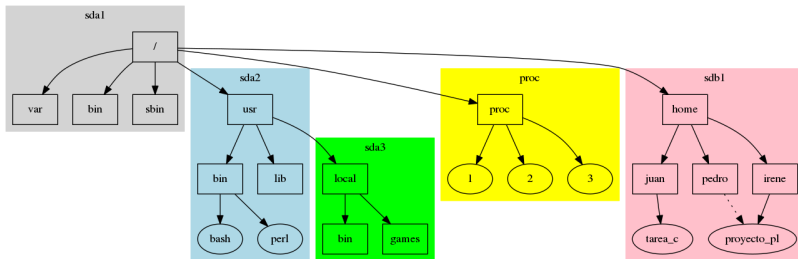


## Unix: Un árbol con *puntos de montaje*

- El sistema *completo* está estructurado en torno a *un sólo árbol*
- Cada sistema de archivos se *monta* en el lugar que el administrador determina
  - El *punto de montaje* es un directorio en el sistema de archivos superior
  - Si ese directorio no está vacío, su contenido *queda oculto* mientras el nuevo sistema de archivos esté montado
    - Salvo implementaciones como UnionFS
- Algunos entornos usuario *abstraen* esta representación, y presentan las unidades *como si fueran una nueva raíz*



# Un árbol con distintos sistemas de archivo *montados*



**Figura:** Árbol formado del montaje de *sda1* en la raíz, *sda2* como `/usr`, *sdb1* como `/home`, y el directorio virtual `proc`



## Windows: sistema de archivos virtual de dos niveles

- La estrategia empleada por Windows es muy distinta
- Cada uno de los sistemas de archivo *detectados* recibe un *identificador de volumen* y es montado automáticamente
- Los usuarios pueden ver a todos los volúmenes montados
  - Nomenclatura VOL:\Dir 1\Dir 2\Archivo.ext
- Dentro de cada uno de los volúmenes encuentran directorios estructurados como árbol
- Algunos volúmenes están preasignados por herencia
  - A y B para unidades de disco flexible
  - C para el disco de inicio del sistema
  - Sigüentes particiones, medios removibles o conexiones sobre red: D, E, F, etc.



# Visión del árbol en un sistema Windows

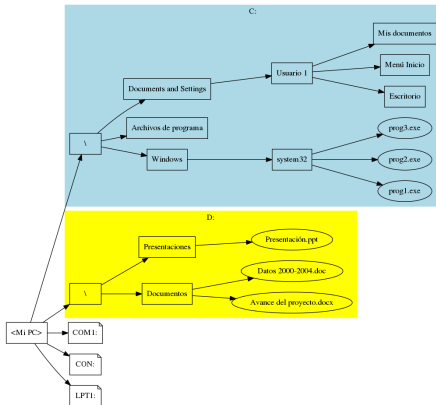


Figura: Vista de un sistema de archivos Windows





# Índice

- 1 Semántica de archivos
- 2 Tipos de archivo
- 3 Métodos de acceso
- 4 Organización de archivos
- 5 **Sistemas de archivos remotos**



# La red y los sistemas de archivos

- *Modelo cliente-servidor*
  - Un equipo que tiene algún recurso que puede ofrecer a otros es un *servidor*
  - Todo equipo que use sus recursos es *cliente*
  - Dos computadoras pueden ser *clientes* y *servidores* mutuos al mismo tiempo
- Uno de los primeros usos de las redes fue compartir archivos
  - De forma *explícita*: Transferencia manual con programas especializados (p.ej. FTP)
  - De forma *implícita* y *automática*: Sistema de archivos sobre la red



## Llamadas a procedimientos remotos (*RPC*)

- Mecanismo de mensajes (y manejo básico de sesión) que causen que una computadora remota ejecute determinada acción
- Pero al mismo tiempo, es un *protocolo específico de descubrimiento de recursos y abstracción* desarrollado en los 1980s por Sun Microsystems
  - Hay muchos protocolos comparables → DCE/RPC (OSF), DCOM (Microsoft)
  - Hoy en día, SOAP y XML-RPC
    - Con mayor independencia de proveedor, pero con mayor complejidad
- Permite al programador delegar en un *servicio* el manejo de las conexiones en red
  - Y limitarse a atender la conexión *ya establecida*



# Network File System (NFS)

- *Network File System* NFS (Unix), construido sobre Sun-RPC (1984)
- Diseñado para aprovechar el hardware existente
  - Un sólo servidor de archivos
  - Todas las estaciones de trabajo *montan* los directorios `/home`
  - Incluso pueden montar otras partes del sistema
  - Estaciones *diskless*



## Semántica básica de NFS

- Implementa la semántica de un sistema local Unix sobre la red
  - Se integra transparentemente en cualquier sistema Unix
- Los sistemas remotos se montan con el mismo comando y lógica que los locales
  - `# mount archivos.unam.mx:/home /home`



# Ventajas de NFS

- Implementación muy sencilla
- Integración semántica directa (para sistemas tipo Unix)
- Protocolo muy ligero
  - Al no hacer verificaciones por operación ni implementar cifrado, es muy conveniente para manejar archivos muy grandes



## Desventajas de NFS

- *Escrituras síncronas* → Seguras, pero lentas
  - Implementaciones posteriores incluyen manejo de caché de escritura
- Manejo de seguridad / credenciales trivial / ingenuo
  - Acceso controlado por dirección IP
  - Requiere correspondencia de nombre-UID de usuarios



# Server Message Block (SMB)

- *Familia* de protocolos de compartición de archivos empleado en sistemas Microsoft (desde ~1990)
- Primeras implementaciones, sobre un protocolo de red propio (NBF, frecuentemente conocido como NetBEUI); migrado gradualmente para operar sobre TCP/IP
  - 100 % sobre TCP/IP desde el 2000
  - También conocido como CIFS (*Common Internet File System*)





# Server Message Block (SMB)

- Se ajusta más cercanamente a la semántica de MS-DOS/Windows
  - *Tiende* a emplearse montando el directorio compartido como una unidad en *Mi PC*:
    - > NET USE W: \\servidor\directorio
  - En Windows, puede referirse a un archivo directamente por su ruta (aunque varios programas no lo implementan)



# Ventajas de SMB

- Hoy en día, implementaciones bastante universales, para cualquier sistema operativo
  - *Samba* → Implementación hecha (originalmente) por *ingeniería inversa*
- Fácil de integrar en entornos basados en Windows
- Si bien las versiones viejas presentan problemas de seguridad, las implementaciones actuales son bastante confiables
  - Particularmente en relación al cifrado de credenciales
- Mecanismo de *bloqueo oportunista* (explicamos pronto)



## Desventajas de SMB

- Protocolo pensado para una red *pequeña*
  - Demasiado *platicador*, puede llevar a congestiones de red
- Resolución de direcciones poco confiable
  - Originalmente basada 100 % en *broadcast*
  - Posteriormente, empleando un servidor *WINS*
  - Actualmente, sobre DNS dinámico
- Semántica de autenticación poco clara
  - Esquemas de autenticación *por recurso*, *por usuario*, basados en directorio
  - Imposibilidad de establecer sesiones múltiples



## Bloqueo oportunista (*OpLock*)

- Bloqueo orientado al *rendimiento*, no a la sincronización
- Busca mejorar el manejo del caché e ignorar (un poco) la semántica de abrir/cerrar archivos

**Para lotes** Si un archivo es abierto y cerrado muchas veces (comportamiento común con los archivos `.BAT` de MS-DOS), este bloqueo *demora* al `close()`, esperando que pronto haya un `open()` al mismo.

**Exclusivos** Una vez otorgado, el cliente asume que es el único manipulando al archivo, y mantener en caché local todos los cambios; el servidor puede *romper* o *revocar* este bloqueo, obligando a una sincronización.



## Sistemas de archivos distribuidos

- Los dos esquemas presentados trabajan con una visión clara cliente/servidor
- Un sistema de archivos *distribuido* busca crear recursos *compartidos*, en que diversos equipos se *repartan* la carga y *compartan* sus recursos
  - Buscando una *fácil escalabilidad*
- Problema complejo: Operar sin la certeza de que todos los nodos estén siempre conectados
- Implementación ejemplo: *Andrew File System* (Carnegie Mellon University, 1989)



# Lógica básica de Andrew File System

- Uso típico en organizaciones grandes (>25,000 estaciones)
- Empleo agresivo de caché
- *Ocultamiento / agnosticismo* a ubicación
  - Un *volumen* puede migrarse a través de una red mientras está siendo empleado
  - Credenciales y ubicación provistos por un servicio *Kerberos*



## Modelo de Consistencia Débil (*Weak Consistency Model*)

- Al abrir un archivo, se copia completo al cliente
  - Lecturas y escrituras se dirigen a la copia local
- Al cerrar el archivo, se copia de vuelta al *servidor de origen*
  - El *servidor de origen* se *compromete* a notificar a los clientes si un archivo abierto fue modificado (*callback*)
- Los cambios a un archivo abierto por un usuario no son visibles de inmediato
- Una vez que se cierra un archivo, los cambios hechos a éste son sólo visibles a sesiones abiertas posteriormente



## Para qué sí es, para qué no

- AFS está pensado para *ciertos* tipos de uso, no para *otros*
- Asume modificaciones a *archivos completos*
  - Es muy *caro* mantener una base de datos compartida
  - No hay bloqueos por rango
- A diferencia de NFS, *no* busca ser universal
  - Necesariamente hay un *espacio local* y un *espacio compartido*
  - No maneja *diskless*

