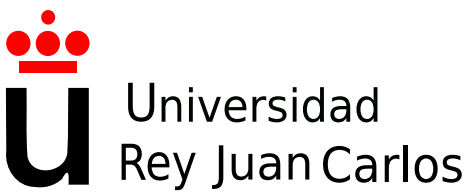
Tema 8: Gestión de ficheros

Enrique Soriano

Laboratorio de Sistemas,
Grupo de Sistemas y Comunicaciones,
URJC

6 de mayo de 2010



Sistema de ficheros en almacenamiento secundario

- Datos que tienen que persistir a la ejecución del proceso y/o del sistema.
- Datos que comparten distintos procesos y/o sistemas.
- Gran cantidad de datos: no caben en memoria.
- Requisitos:
  - Gran tamaño
  - Durabilidad
  - Acceso concurrente
  - Protección

Ficheros

- Estructura: fichero simple (UNIX), Resource forks (Mac)...
- Nombre: ¿extensión?
- ¿Tipos?

Estructura interna los ficheros

- Bloque físico $\longrightarrow$ determinado por el disco (sector).
- Bloque lógico $\longrightarrow$ determinado por el sistema de ficheros.
- Si no coinciden, empaquetamos.
- Siempre hay fragmentación interna en los bloques.
- ¿Tamaño? compromiso entre:

$$\uparrow \text{tamaño de bloque} \Rightarrow \uparrow \text{fragmentación}$$
$$\uparrow \text{tamaño de bloque} \Rightarrow \uparrow \text{tasa de transferencia}$$

Formas de acceso

- Secuencial: se accede registro a registro (contiguos). Antiguo (cintas magnéticas).
- Aleatorio: se accede a cualquier registro inmediatamente. Los HD permiten este acceso.

Particiones (PC)

- MBR: Primer bloque del disco, 512 bytes.
  - Cargador primario: 440 bytes.
  - Tabla de particiones primarias: 4 entradas de 16 bytes.
    - Arrancable/no arrancable.
    - Dirección del primer bloque (CHS).
    - Tipo de partición (ntfs, linux, linux swap, plan9, ...).
    - Dirección del último bloque (CHS).
    - Dirección del primer bloque (LBA).
    - Número de bloques de la partición.
- Partición lógica: partición dentro de una partición primaria extendida.

Asignación de espacio

- ¿Cómo repartir los sectores de la partición entre los ficheros del sistema de ficheros?

Asignación contigua

- Un fichero ocupa una serie de bloques contiguos en disco.
- Acceso a disco rápido.
- Rendimiento alto: un acceso para conseguir un dato cualquiera.
- Problema: asignación dinámica $\longrightarrow$ fragmentación externa.
- Problema: hay que saber el tamaño máximo del fichero cuando se crea.
- Compactación.

Asignación enlazada

- Un fichero es una lista enlazada de bloques.
- Se tienen punteros al primer y al último bloque.
- Datos por bloque = tamaño de bloque - tamaño de estructura.
- Ventajas:
  - No hay fragmentación externa.
  - Tamaños no limitados por huecos.
- Problemas:
  - Acceso aleatorio ineficaz.
  - En discos duros tradicionales: la cabeza va y viene.
  - Se desperdicia una estructura por bloque.

¿qué pasa si se pierde una estructura?

Asignación enlazada de clusters

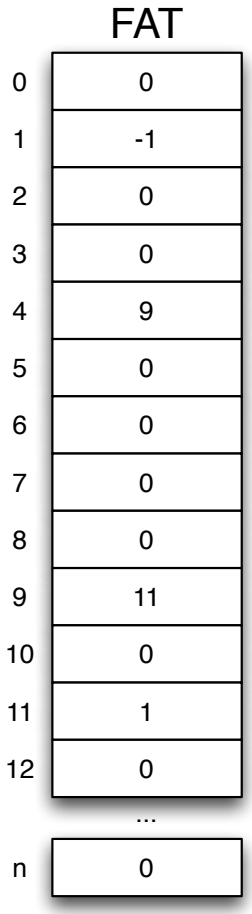
- Cluster: agrupación de bloques contiguos (e.g. 32 Kb).
- No se desperdicia tanto (1 estructura para n bloques).
- Búsqueda más rápida, menos saltos en la lista.
- Aumenta la fragmentación interna.

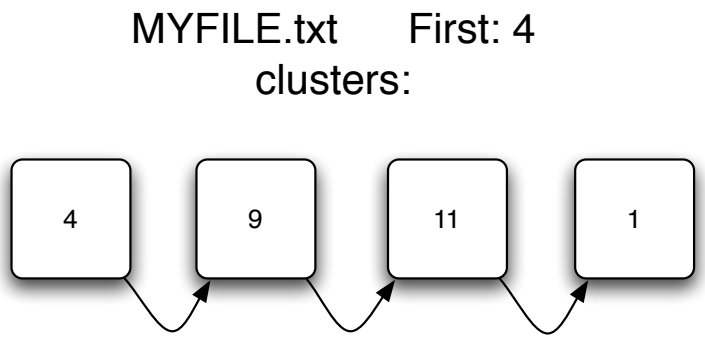
Asignación enlazada con tabla: FAT

Tabla FAT:

- Una entrada por cluster.
- El índice de la tabla es el número de cluster.
- La entrada de directorio tiene la referencia a la entrada primer cluster.
- La entrada del cluster actual referencia la entrada del siguiente cluster.
- Mejora del acceso aleatorio.

Asignación enlazada con tabla: FAT





Asignación enlazada con tabla: FAT

- El acceso aleatorio es más rápido que en lista enlazada normal.
- Tabla FAT en memoria principal $\longrightarrow$ no puede ser muy grande $\longrightarrow$ clusters grandes.
- ¿Y si se estropea la tabla FAT?

Partición FAT32

- El sector 0 de la partición es el *boot sector* (también está duplicado en el sector 6). Contiene código del cargador e información sobre el sistema de ficheros:
  - Nº de sectores, 4 bytes (Max. tam. de disco: 2 Tb)
  - Etiqueta del volumen.
  - Nº de copias de la tabla FAT.
  - Primer cluster del ráiz.
  - ...

FAT32

- Entrada de directorio (32 bytes):
  - Nombre del archivo, 8 Bytes.
  - Extensión del archivo, 3 Bytes.
  - Atributos del archivo, 1 Byte.
  - Reservado, 10 Bytes.
  - Hora de la última modificación, 2 bytes.
  - Fecha de la última modificación, 2 bytes.
  - Primer cluster del archivo, 4 bytes.
  - Tamaño del archivo, 4 bytes.
- Clusters de 32 Kb, 64 sectores.

Asignación indexada

- Se colocan todos los punteros a bloque juntos $\longrightarrow$ acceso aleatorio más rápido.
- Cada fichero tiene un bloque de indirección con los punteros a sus bloques de datos.

Asignación indexada

- Bloque de indirección grande $\longrightarrow$ desperdicio.
- Bloque de indirección pequeño $\longrightarrow$ no soporta ficheros grandes.
- Para un acceso a datos, siempre son dos accesos a disco.

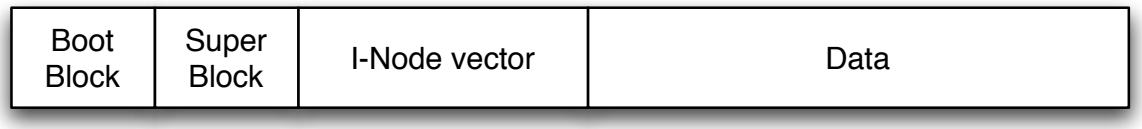
Asignación indexada: multinivel

- Los bloques de índice de 1er nivel apuntan a bloques de índice de 2º nivel.
- Los bloques de índice de 2º nivel apuntan a bloques de datos.
- Ejemplo: bloques de 4Kb con punteros de 4 bytes.
- Problema: Para los ficheros pequeños, desperdiciamos.
- Problema: Para un acceso a datos, n accesos a disco.

Asignación indexada: esquema combinado

- Algunos bloques directos de datos.
- Algunos bloques de indirección simple.
- Algunos bloques de indirección doble.
- ...
- Nos quedamos con lo mejor de cada método anterior.

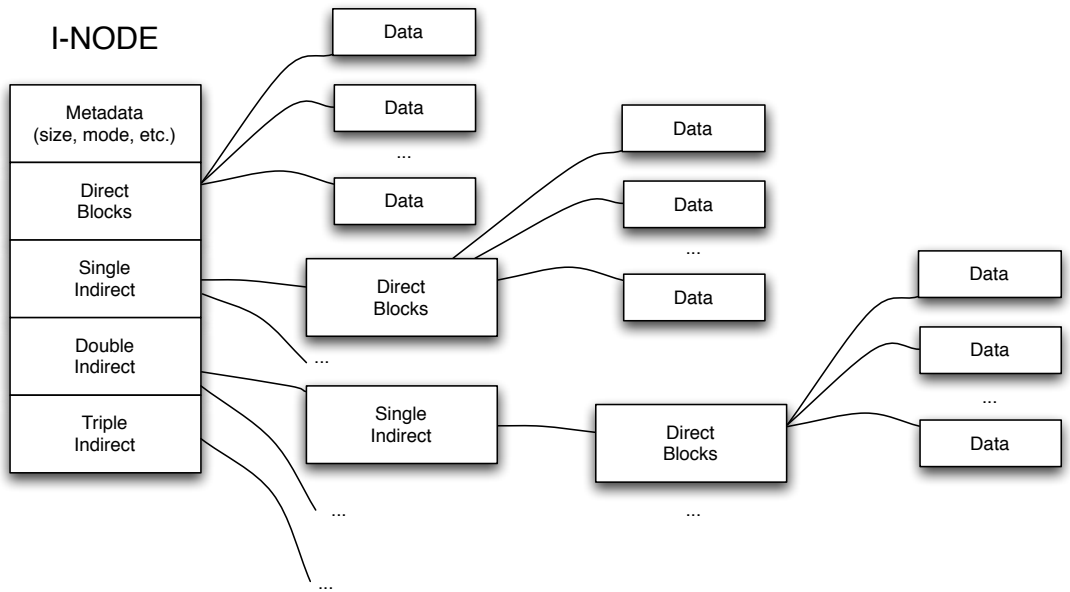
Partición UNIX



- Bloque de arranque (boot block): código de arranque del sistema.
- Super bloque: tamaño del volumen, número de bloques libres, lista de bloques libres, tamaño del vector de i-nodos, siguiente i-nodo libre, cierres...
- Vector de i-nodos: representación de los ficheros.
- Bloques de datos.

Ficheros en UNIX: I-NODOS

Asignación indexada con esquema combinado:



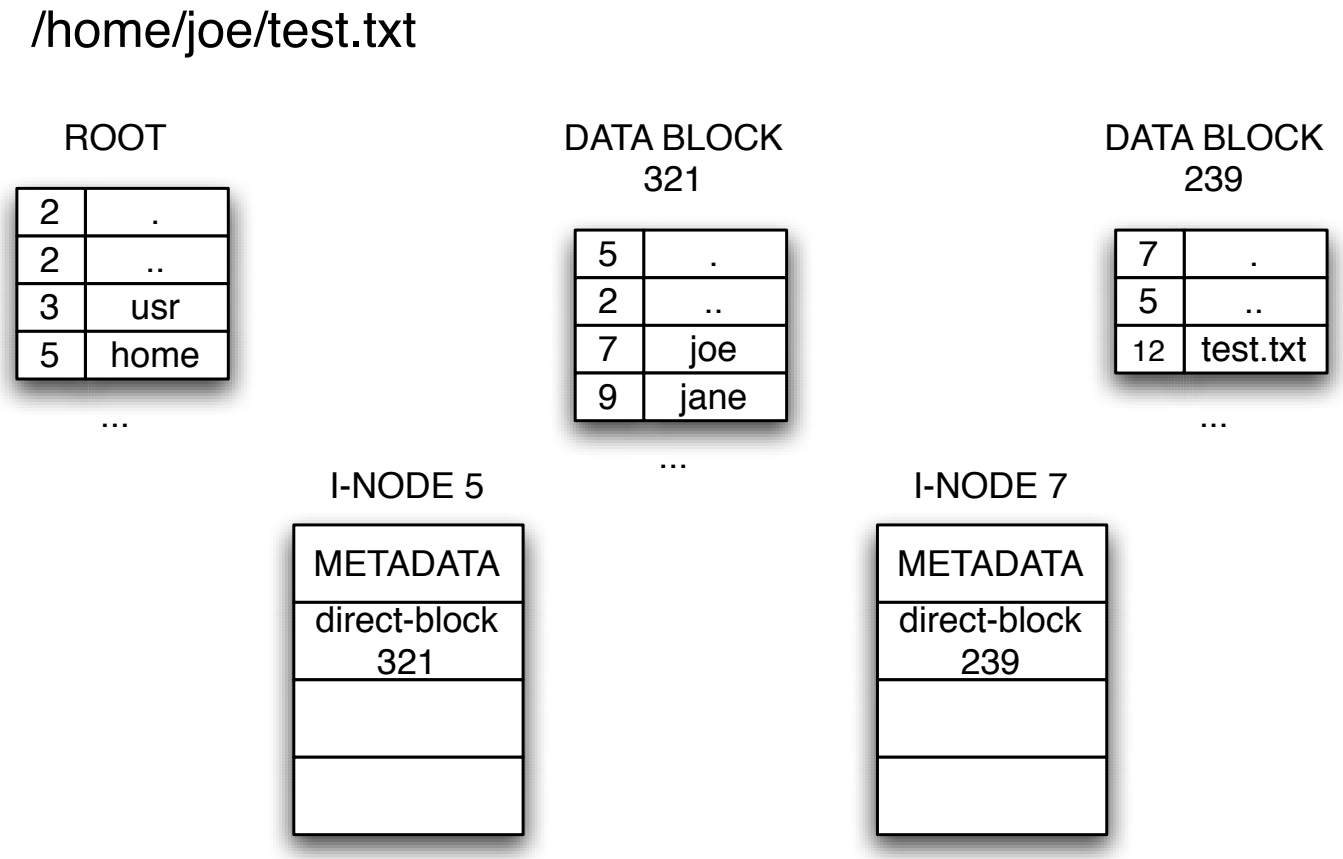
Ficheros en UNIX: I-NODOS

- Cada fichero/directorio tiene un i-nodo asociado, definido por un número de i-nodo. El directorio raíz siempre tiene el número de i-nodo 2.
- La estructura se localiza en el vector indexando por el número de i-nodo.
- La estructura contiene:
  - permisos
  - tiempos
    - acceso a datos (`atime`)
    - modificación de los datos (`mtime`)
    - modificación del i-nodo (`ctime`)
  - tamaño
  - dueño
  - tipo
  - número de bloques
  - contador de referencias (links)
- **No** contiene el nombre de fichero.

Directorios en UNIX: I-NODOS

- Un directorio relaciona un nombre con un i-nodo, en una entrada.
- Tiene:
  - Su propio i-nodo
  - Bloques de datos con la lista de entradas
- Entre las entradas, tiene la entrada de . (su i-nodo) y .. (i-nodo del padre).

Ficheros en UNIX: I-NODOS



Implementación de directorios

- Lineal
  - Pro: simple.
  - Contra: búsqueda lineal.
- Lineal ordenada
  - Pro: búsqueda binaria.
  - Contra: complica creación y borrado.
- Tabla hash con desbordamiento
  - Pro: mucho más rápida que lineal.
  - Contra: más complejidad.

Gestión de espacio libre

- Bitmap: rápido.
- Lista enlazada de bloques libres: ineficiente si buscamos varios, pero por lo general sólo se busca uno libre.
- Agrupaciones: el primer nodo en la lista no está vacío; tiene enlaces a n bloques libres, los n-1 primeros vacíos, el último con n más.
- Cuenta: guarda en una lista el bloque libre y el número de bloques contiguos libres que le siguen.

Espacio de nombres

- Contexto en el que se traduce un nombre al fichero correspondiente.
- En Plan 9, ¿es el mismo `/dev/cons` en una ventana que en otra?

Espacio de nombres

- En Plan 9 cada proceso puede tener su propio espacio de nombres: cada proceso puede tener su propia **tabla de montaje**.
- En Plan 9 por defecto el hijo lo comparte con el padre (se puede copiar o empezar desde cero: `rfork`).
- En UNIX hay una tabla de montaje para toda la máquina (/etc/fstab).

Espacio de nombres

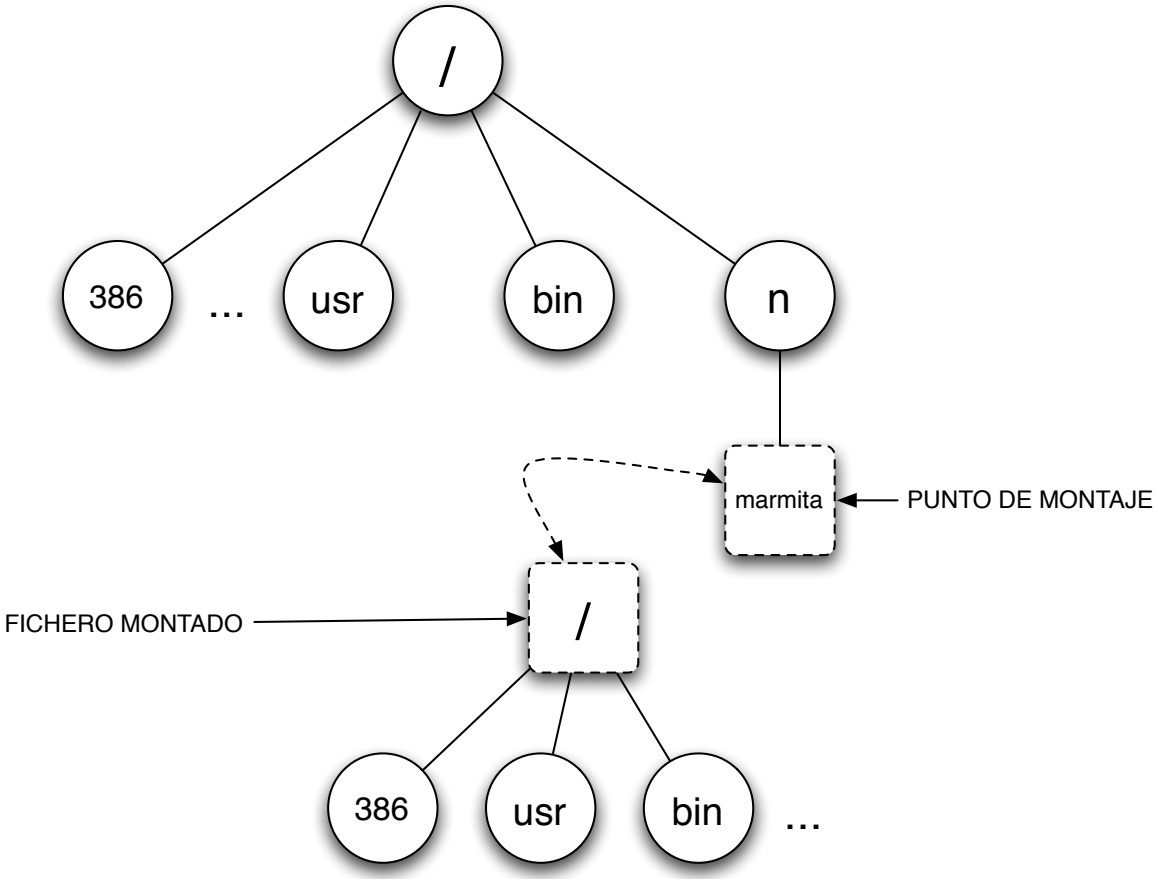
- Se “camina” (walk) por la ruta, resolviendo cada parte.
- Se pueden “pegar” árboles a nuestro espacio de nombres: `mount`.

Espacio de nombres

- Ejemplo en Plan 9: montar un servidor de ficheros en red:

```
term% srv tcp!marmita.lsub.org!9fs
term% mount -c /srv/tcp!marmita.lsub.org!9fs
/n/marmita
term% cd /n/marmita ; ls
term% cat /proc/$pid/ns
temr% ns
```

Espacio de nombres



Imágenes de disco

- Un fichero que tiene dentro la estructura completa sector por sector de un disco duro, CD-ROM (ISO 9660), floppy, etc.
- Útil para clonar sistemas, backups, crear CD/DVD, descargar discos, máquinas virtuales, etc.

Imágenes de disco

- Ejemplo en Linux: crear y usar una imagen de floppy:

```
~$ # creo un fichero de 1,44 MB con ceros
~$ dd if=/dev/zero of=/tmp/f bs=512 count=2880
~$ # formateo la imagen de floppy con VFAT
~$ mkfs.vfat -v -c /tmp/f
~$ # monto en el punto de montaje /mnt/floppy
~$ mount -o loop /tmp/f -t vfat /mnt/floppy
~$ # creo un fichero en el floppy
~$ touch /mnt/floppy/afile
~$ # ver el espacio de nombres
~$ mount
```

Uniones

- Consiste en *unir* varios directorios en uno.
- Utilidades: compartir código fuente, agrupar binarios (/bin), reunir ficheros para crear un backup (CD,DVD,...), distribuciones *live*, etc.
- Idea de Plan 9. UnionFS y Aufs en UNIX.

Uniones

- Ejemplo en Plan 9: crear una union de tres directorios en el directorio `$home/union`

```
term% bind -bc /tmp/a $home/union
term% bind -bc /tmp/b $home/union
term% # en $home/union vemos los ficheros de
term% # /tmp/b, /tmp/a, y $/home/union, en ese orden
term% ls $home/union
```

Enlaces

- En Plan 9 no hacen falta. En UNIX sí.
- Enlaces duros: es otro nombre para el fichero, la entrada de directorio apunta al mismo i-nodo que el antiguo nombre.
- Enlaces simbólicos: es un fichero cuyos datos contienen la ruta al fichero enlazado $\longrightarrow$ pueden *romperse*.

Proyección en memoria

- mmap: *mapea* un fichero en memoria para poder acceder a los datos del fichero como si fuese un *buffer*, sin usar `read` ni `write`.
- El fichero se trae poco a poco $\rightarrow$ paginación por demanda.
- Llamadas al sistema: `mmap` y `munmap` en UNIX.

Cache

Dos aproximaciones:

- Buffer cache: parte de la memoria se usa para almacenar bloques.
- Page cache: aprovecha los marcos de página no usados por procesos para guardar los datos de los ficheros (Unified Virtual Memory). El fichero se trae en demanda.
- En ambos casos, es transparente para el usuario.

Cache

Aquí sí podemos implementar LRU para reemplazar PERO:

- ¿Es probable que el último se vuelva a usar?
  Ejemplo: último bloque leído en un acceso secuencial.
- ¿Influye en la coherencia del sistema?
  Ejemplo: un bloque de indirección que se acaba de escribir debería ir directamente al disco.
  Ejemplo: ¿y los datos de un directorio?

Cache

Escrituras:

- Síncrona (write-trough): evita problemas de coherencia.
- Asíncrona (delayed write-back): incrementa el rendimiento.

Cache

Para accesos secuenciales:

- Read-ahead.
- Free-behind.

Recuperación

- Programas para reparar el sistema de ficheros, p. ej. `fsck` en UNIX.
- Hay que recorrer los bloques de los ficheros para detectar errores, p. ej.:
  - Un bloque en un fichero y en la lista de libres a la vez.
  - Un bloque en dos ficheros a la vez.
- Esto es caro.

Journaling

Ideas básicas:

- Las operaciones se anotan en un *journal* de forma **atómica**.
- Una vez escritas en el *journal* las operaciones están comprometidas.
- En algún momento las operaciones se aplicarán en el FS.
- Si el sistema se queda *colgado*, al rearrancar se aplican todas las operaciones pendientes en el *journal* (si hay) para recuperar la coherencia.
- Desventaja: hay que escribir los datos dos veces en el disco.

¿Cómo está nuestro servidor?

- Fossil es un sistema de ficheros en área de usuario.
- Fossil tiene *snapshots*.
- Venti es un servidor de bloques indexados por hash SHA-1 (20 bytes).
- Venti nunca borra los bloques.