Tema 6: Sistemas de ficheros y discos

Programación y Administración de Sistemas (2022-2023)

Javier Sánchez Monedero

13 de abril de 2023

Tabla de contenidos

1 Asignación de bloques

Asignación de bloques

Asignación: cómo se hace la correspondencia entre los bloques físicos del disco y los bloques lógicos del archivo.

Mecanismos de asignación: Asignación de **bloques contiguos**:

- Todos los bloques del archivo se encuentran contiguos en el disco:
  - Muy sencillo de implementar. ✔
  - Accesos secuencial y directo muy rápidos. ✔
  - Necesario saber el tamaño del archivo al crearlo. ✘
  - Fragmentación del disco. ✘
  - Para añadir datos al archivo, puede que haya que moverlo. ✘
- Por todo ello, no se utiliza.

Mecanismos de asignación

Mecanismos de asignación: Asignación de **bloques no contiguos**:

- Los bloques del archivo se encuentran en cualquier posición del disco.
  - Se produce menos **fragmentación** → el primer bloque asignado es el primero que hay libre. ✔
  - Es necesario traducir el número de bloque lógico al número de bloque en el dispositivo. ✖
- Es la opción utilizada en la mayoría de SOs.

Para tener constancia de qué bloques no contiguos pertenecen a cada archivo, se utilizan **listas enlazadas** o **índices** (que pueden ser multinivel).

- ISO9660: Inicio y tamaño (fichero contiguo).
- SF MS-DOS: FAT (fichero enlazado).
- SF UNIX: i-nodo (fichero indexado).
- NTFS: Registro Windows (fichero indexado).

Lista enlazada

Lista enlazada:

- Cada bloque tiene un apuntador al siguiente bloque que seguiría en el archivo.
- El descriptor del archivo solo debe incluir la referencia al primer bloque.
  - El acceso secuencial es muy rápido. ✔
  - El acceso aleatorio a un bloque concreto de un archivo es muy costoso. ✖
  - Cada bloque incluye un apuntador que aumenta su tamaño (y complica el cálculo de espacio libre). ✖
  - La pérdida de un bloque supone perder el archivo completo. ✖

Tabla de asignación de archivos (FAT)

Tabla de asignación de archivos:

- Es una variación del **método lista enlazada**.
- Los apuntadores se almacenan en una tabla independiente de los bloques (*File Allocation Table*, FAT).
- La tabla posee una entrada por cada bloque del SA.
- La FAT ocupará un espacio prefijado en la partición.
- Descriptor fichero → incluye su primera posición en la tabla.
- Acceso aleatorio al archivo: recorriendo la tabla.
- La tabla se aloja en caché para mejorar las prestaciones y se mantiene una copia doble en el disco para mayor fiabilidad. ¡La FAT puede llegar a ocupar mucho! → agrupaciones.

Tabla de asignación de archivos (FAT)

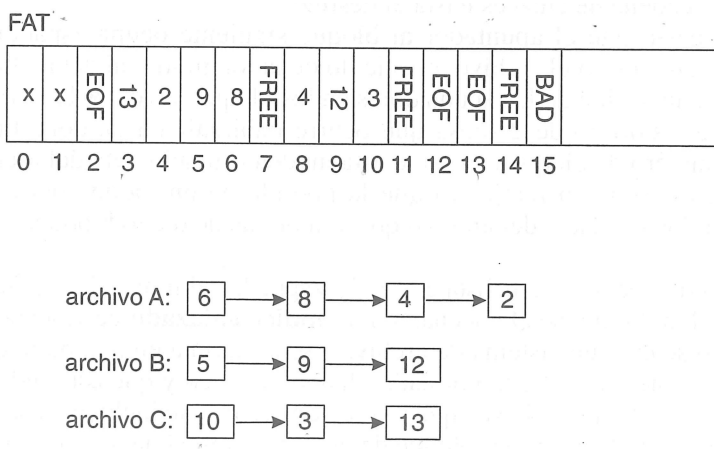
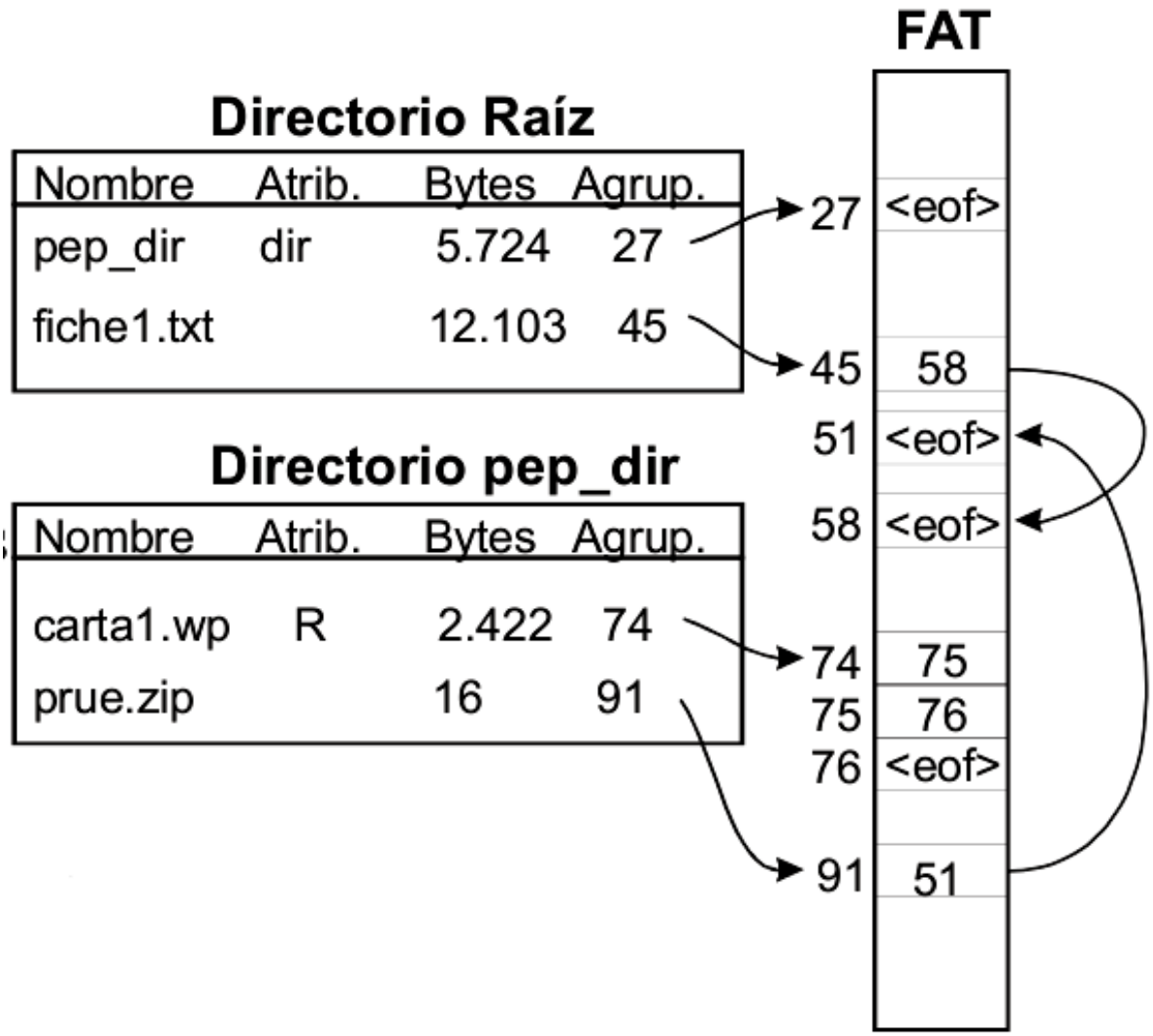


Tabla de asignación de archivos (FAT)

- FAT de 12 bits: 4K agrupaciones.
- FAT de 16 bits: 64K agrupaciones.
- FAT de 32 bits: 2^{28} agrupaciones (solo usa 28 bits). Tamaño de fichero en directorios → 32 bits. Tamaño máximo $2^{32} - 1 = 4GB - 1$



Índices

- Los punteros a los bloques están juntos y contiguos en una localización concreta → **Bloques índice**.
- Cada archivo tiene un bloque índice.
- Para buscar el i-ésimo bloque de un fichero, buscamos la i-ésima entrada en su bloque índice.
  - Buen acceso directo. ✔
  - Se evita la fragmentación. ✔

  - ¿Tamaño del bloque índice? → debe fijarse un número de entradas y hay que reservar espacio para todas ellas. ✖
  - Limitamos el tamaño máximo de los archivos. ✖

Índice multinivel

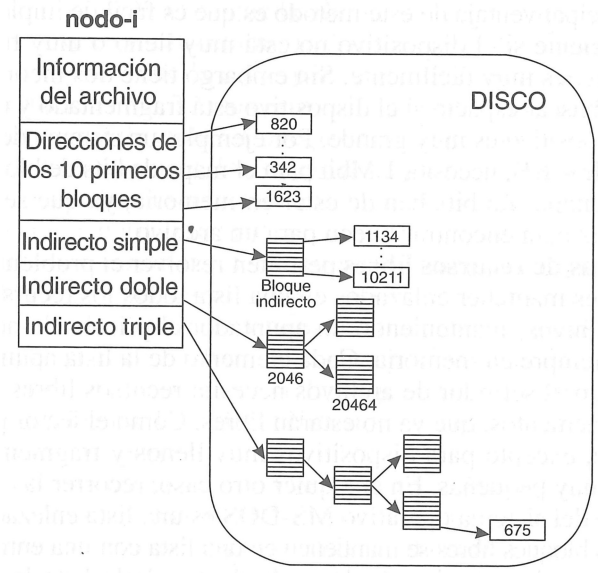
- Consiste en introducir n **niveles de apuntadores**, de manera que los apuntadores del descriptor apuntan a otros.
- Índice multinivel de nivel 1: el bloque índice apunta a otros bloques índices que finalmente apunta a un bloque de datos del fichero.
- Evita tener que prefijar el tamaño del bloque índice (podemos poner apuntadores a NULL). ✔
- El bloque índice tendrá un número pequeño de entradas. ✔
- Cada nivel, supone un acceso a disco adicional. ✖
- Para archivos pequeños, se desaprovechan muchos bloques índice. ✖

Esquema híbrido: nodos-i

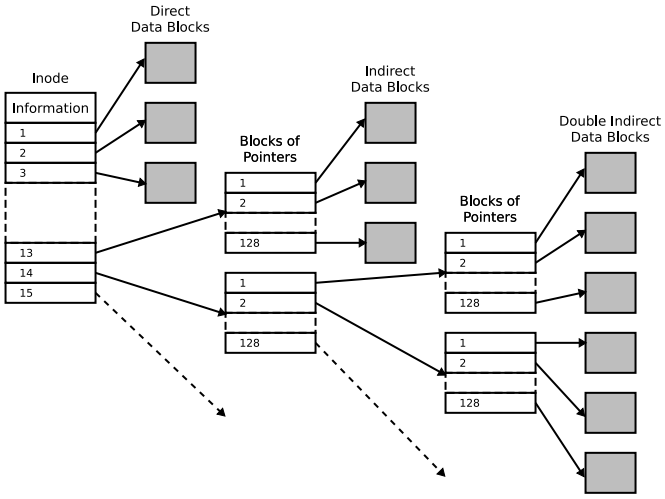
Solución UNIX → Esquema híbrido

- Por cada nodo-i incluir 15 punteros:
  - Punteros directos a los 12 primeros bloques (para archivos pequeños). Nota: la siguiente figura indica 10 erróneamente.
  - Puntero a un bloque índice de primer nivel (donde encontraremos punteros a bloques).
  - Puntero a un bloque índice de segundo nivel (donde encontraremos punteros a punteros a bloques).
  - Puntero a un bloque índice de tercer nivel (donde encontraremos punteros a punteros a punteros a bloques).
- Bloques del disco: bloques de datos o bloques índice.
- En **ext4** y en **NTFS** existen los ***extents*** (bloques índice especiales que marcan una zona contigua del disco "numeroBloqueInicial, numeroBloques").

Asignación de bloques: indexado multinivel



Asignación de bloques: indexado multinivel Ext2



2 Gestión del espacio libre

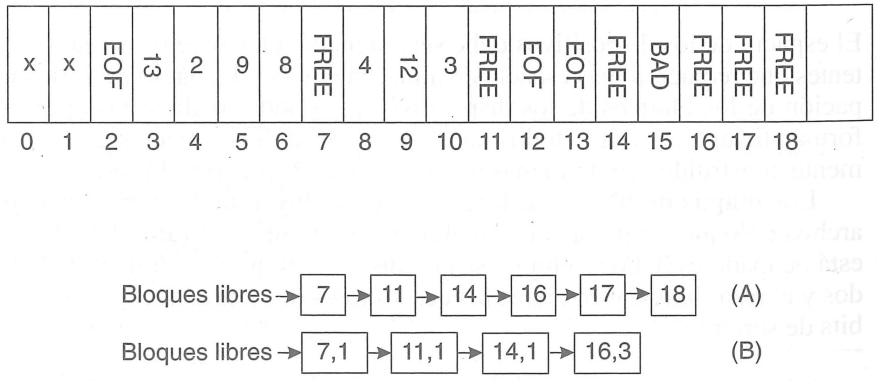
Gestión del espacio libre

- **Gestión del espacio libre**: se necesita para asignar espacio a los archivos nuevos o a los que se les desea añadir datos.
- Se mantienen **mapas de recursos**, implementados como mapas de bits o listas de recursos libres.
  - **Mapas de bits**
    - Se incluye un bit por recurso (descriptor de archivo, bloque o agrupación), que será 1 si el recurso esta libre y 0 en caso contrario. Muy sencillo de implementar y de usar. Disco poco fragmentado → bloques libres al final, búsqueda muy eficiente. Disco fragmentado → búsqueda más costosa. Espacio adicional requerido por el mapa.
    - FAT: la propia tabla actúa como mapa de recursos.

Gestión del espacio libre

- **Gestión del espacio libre**.
  - **Listas de recursos libres**
    - Mantener una lista de apuntadores a los recursos libres.
    - Al ocupar un recurso lo borramos de la lista. Muy eficiente para discos muy llenos y fragmentados. Disco con mucho espacio libre → Ineficiente debido a que hay que cargar la lista.
    - Solución → Incluir número de bloques libres consecutivos en la lista.

Gestión del espacio libre: FAT + lista enlazada



3 Incremento de prestaciones

Incremento de prestaciones

- Acceso a **memoria** → orden de nanosegundos.
- Acceso a **disco** → orden de milisegundos.
  - **Almacenamiento intermedio de los datos**
    - Mantener una caché de datos en Memoria Principal (MP).
    - Aprovecha la proximidad espacial y temporal en las referencias a los datos accedidos.
    - **Caché de nombres**: lista con `{nombre,nodo-i}`. Si se vuelve a acceder al archivo, no hay que hacer toda la búsqueda del `nodo-i`.
    - **Caché de bloques**: colección de bloques leídos o escritos recientemente. Si se vuelve a acceder a ese bloque, no hay que cargarlo de nuevo.

Incremento de prestaciones

- **Caché de bloques**:
  - Si el bloque está en MP, se escribirá o leerá en MP.
  - Posteriormente, se moverán los bloques de MP al dispositivo.
  - Si la caché está llena, hay que eliminar algún bloque:

    * **Políticas de reemplazo**: *First In First Out* (FIFO), *Most Recently Used* (MRU), *Least Recently Used* (LRU)...
    * Lo más común es **LRU**: aprovecha que los bloques no utilizados durante mucho tiempo, posiblemente no volverán a ser utilizados. Peligroso si hay un fallo del SA.

Incremento de prestaciones

- **Caché de bloques**:
  - Bloques sucios (cambiados en caché pero no en el disco). Distintas políticas a la hora de mantener la coherencia:
    * **Escritura inmediata** (*write-through*) → Siempre actualizado.
    * **Escritura diferida** (*write-back*) → Actualizamos cuando el bloque salga de la caché.
    * **Escritura periódica** (*delayed-write*) → Establecer un tiempo periódico para las actualizaciones. Compromiso entre rendimiento y fiabilidad. Reduce la extensión de los posibles daños por caídas.
  - Se puede distinguir entre bloques **especiales** (directorios, nodos-i o bloques índice) y bloques de **datos**. Bloques especiales → *write-through*.
  - No se debe quitar un disco del sistema sin antes volcar los datos de la cache (comando `sync`).