Bloque II – Sistemas Operativos

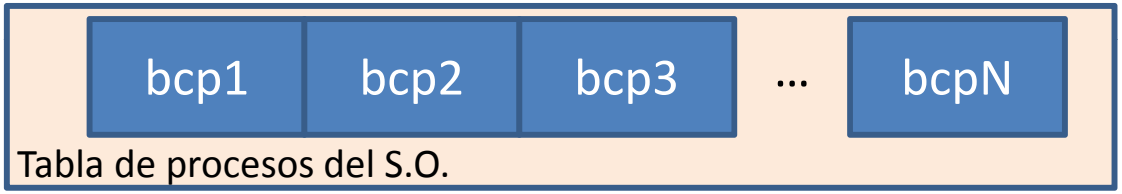
3. Procesos. Introducción

Procesos y programas

- Programa Vs proceso:
  - Un **programa** se puede ver como una entidad **pasiva**. Se puede ver como código que reside en disco o memoria y que no ha sido activado.
  - La activación de este código, implica:
    - Su carga en memoria (todo o parte de él).
    - La creación (por parte del S.O.) de estructuras que permitan controlar su ejecución. Estas estructuras constituyen el BCP (bloque de control del proceso).
    - Un proceso es por tanto una entidad **activa**.
    - Un proceso se puede ver como una instancia de un programa en ejecución.
    - *P.ej. Un usuario puede abrir 2 veces el firefox, esto da lugar a que haya 2 procesos ejecutando el mismo código (aunque con estados diferentes)*
  - Un **proceso** es pues un programa en ejecución, incluyendo el código o instrucciones que lo componen, el contador de programa, los registros y las variables, es decir, contiene toda la información relativa al entorno en donde se ejecuta.

Procesos y programas

- Todo programa que se ejecute lo hace en modo de un **proceso**.
- En general, **proceso** se puede considerar como:
  - la unidad de procesamiento gestionada por el S.O.
- Para ejecutar un programa, éste (su código máquina) ha de residir en el mapa de memoria (*memoria que puede direccionar*), junto con sus datos.
- El S.O. ha de conocer, para cada proceso:
  - Su estado, características y recursos utilizados
    - Esta información de cada proceso se guarda en un BCP (Bloque de control de proceso)
  - El S.O. guarda una Tabla de procesos que contiene 1 BCP x proceso.



Bloque de Control de Proceso (BCP)

- Contiene la info básica de cada proceso.
  - **Información de identificación**.
    - Identificador del proceso.
    - Identificador del proceso padre en caso de existir relaciones padre-hijo (UNIX).
    - Información sobre el usuario (identificador de usuario, grupo)
  - **Estado del procesador** (registros, PC, …)
  - **Información de control del proceso.**
    - Información de planificación y estado:
      - Estado del proceso: (en ejecución, listo, parado,…)
      - Evento por el que espera (si está bloqueado)
      - Prioridad del proceso
      - Información de planificación (p.ej. Uso cpu, time-stamp uso,…)
    - Info de los segmentos de memoria asignados al proceso
      - Punteros a:
        - » segmento de datos:
          - Con/sin valor inicial (valores iniciales del programa)
          - dinámicos (se crean o destruyen durante la ejecución del programa)
        - » segmento de código (instrucciones máquina del programa)
        - » segmento de pila.
    - Recursos asignados al proceso:
      - Ficheros abiertos (tabla de descriptores)
      - Puertos de comunicación asignados
      - Temporizadores
  - **Punteros para estructurar los procesos** (en colas, anillos, …)
  - **Comunicación** entre procesos: El BCP puede contener espacio para almacenar: Las señales y los mensajes enviados al proceso.

Vida de un proceso

- **Creación → ejecución (interrup/activación)→ terminación/muerte**
- Creación
  - La realiza un proceso (padre), salvo el proceso INIT (creado en el arranque: Unix/linux)
  - **Consiste en** completar todas las informaciones del proceso
    - Asignar espacio de memoria para su "mapa de memoria" (todas las regiones de memoria accesibles para él)
    - Seleccionar un BCP libre en la tabla de procesos
    - Rellenar el BCP con la info del proceso (identificación, regiones de memoria, valores iniciales,…)
    - Cargar:
      - En segmento de código sus instrucciones + rutinas del sistema que usará.
      - En el segmento de datos, los datos iniciales del programa (inicializados).
    - Crear la pila inicial del proceso.
    - Marcarlo como **listo** para ejecutarse.→ el planificador lo tendrá en cuenta, cuando lo considere oportuno.

Vida de un proceso

Creación de procesos (fork)

- Fork() … *unix/linux*

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

main()
{
  int fpid,val=2;
  fpid=fork();  //se crea 1 hijo que es copia exacta.
  if (fpid==-1) //la ejecución continúa aquí (padre e hijo)
    printf(“\n fork ha fallado”); /*error */
  else if (fpid==0)
    val--;          /*proceso hijo*/
  else
    val++;          /*proceso padre */

  printf("\n val= %d\n",val);
}
```

Vida de un proceso

interrupción / activación

- Interrupción de un proceso.
  - Un proceso puede ver interrumpida su ejecución (pasar a estar bloqueado o en espera) debido a una interrupción → pasará a ejecutarse el S.O. (modo *kernel*)
  - Es necesario guardar el contenido de:
    - Los registros de usuario en el **BCP**.

- Activación de un proceso.
  - Activar un proceso consiste en ponerlo en ejecución.
  - La realiza un módulo del S.O. llamado "activador" o "*dispatcher*".
  - Cuando se reactive el proceso, la cargarán los valores de los registros (almacenados en el **BCP**) de nuevo en la CPU (incluido el PC), y se continuará normalmente.

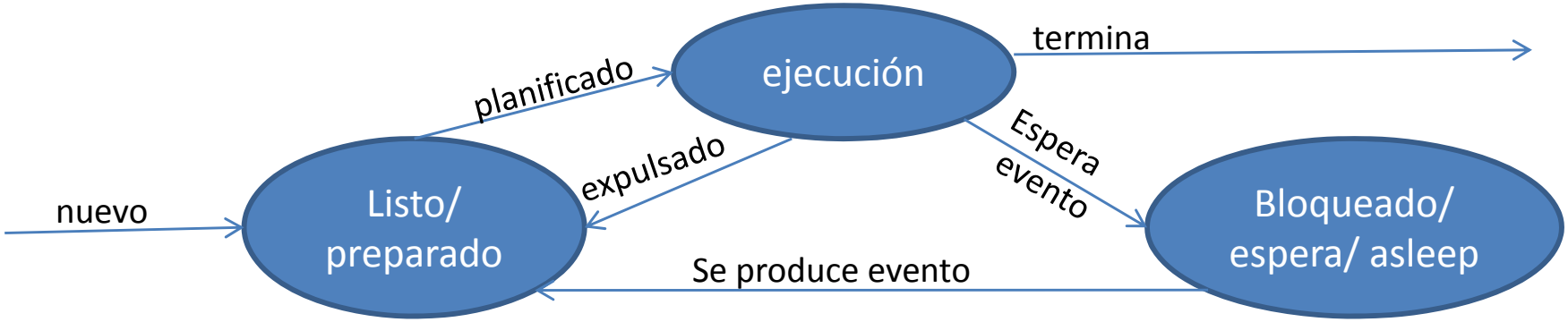
    Esto es, se deja el computador exactamente tal y como se estaba antes de que el proceso dejase la CPU.

Vida de un proceso

- Terminación de un proceso.
  - Motivos:
    - Terminación normal (exit en Unix)
    - Terminación por error (p.ej. Fichero no existe)
    - Error fatal (excepción, acceso a memoria no permitida,…)
    - Terminado por otro proceso (kill -9 Pid)
  - El S.O. debe recuperar los recursos que pudiese tener asignados.
    - Recuperar recursos asignados exclusivamente al proceso (p.ej: BCP, o una región de datos no compartidos).
    - Recuperar recursos compartidos: El S.O. sólo recupera el recurso cuando deja de estar usado (contador de referencias = 0)

Estados de un proceso: activos

- Estados activos: El proceso compite por la cpu o puede hacerlo.
  - Ejecución: El proceso tiene el control de la cpu. El estado del procesador reside en sus registros (registros de la cpu)… BCP puede tener copia sucia.
  - Preparado o listo: Está preparado para acceder a la CPU cuando el planificador lo considere oportuno. El estado reside en el BCP.
  - Bloqueado o espera: Ante una operación que requiera tiempo largo (p.ej. una E/S), el proceso queda bloqueado. El estado del procesador se guarda en el BCP (otro proceso diferente podría ejecutarse mientras tanto).



- Estados inactivos: Procesos no pueden competir por el procesador.
  (1)Suspendido bloqueado, (2)Suspendido listo

Estados de un proceso: inactivos (suspendidos)

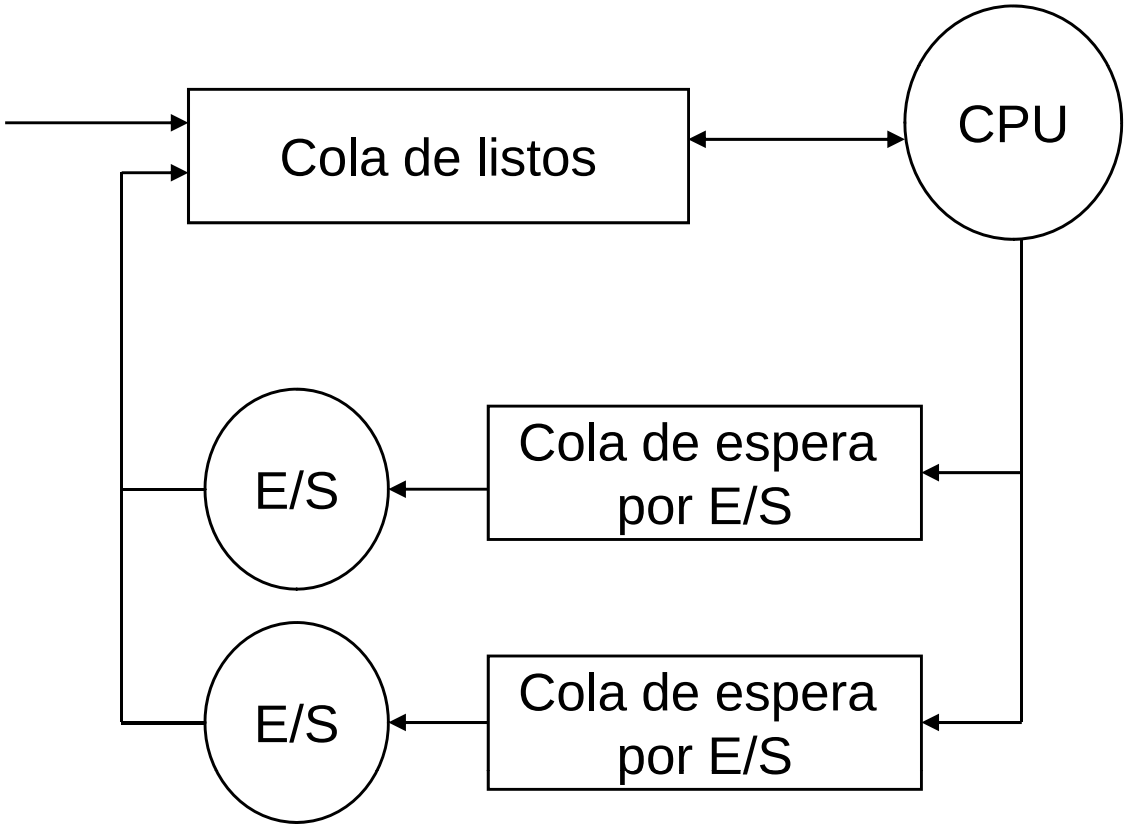
- Estados activos: El proceso compite por la cpu o puede hacerlo.
  - Ejecución, listo, bloqueado
- Estados inactivos: Procesos no pueden competir por el procesador.
  - Para **aumentar el grado de multiprogramación (número de procesos activos en un momento dado)** el S.O. puede **suspender** algunos procesos. Les retira sus marcos de página y aumenta así la memoria disponible para otros procesos.
    - Los marcos retirados se van a disco: a la zona de *intercambio.*
  - **Suspendido y listo**: Estaba listo, y lo suspenden.
  - **Suspendido y bloqueado**: Estaba bloqueado (esperando por un evento) y lo suspenden.

Estados de un proceso:
motivos de transiciones de estado

- Inicio de proceso → inserción en cola de listos.
- Paso a ejecución → se saca el primero de cola de listos
- Paso a bloqueado → Un proceso
  - Realiza una operación de E/S y debe esperar
  - Estaba "suspendido y bloqueado" y cuando le toca reanudarse todavía no ha desaparecido la causa de su bloqueo (espera).
- Paso a listo
  - Ejecución del programa (inicio)
  - Fin de operación de E/S
  - Interrupción que fuerza que se quite de la CPU a un proceso (p.ej fin quantum)
  - Activación: Un proceso estaba suspendido y listo y se saca de estado de suspensión.
- Paso a suspendido bloqueado.
  - Estaba bloqueado y el S.O. da la orden de suspenderlo.
- Paso a suspendido listo.
  - Suspensión de un proceso listo.
  - Desbloqueo de un proceso suspendido y bloqueado (p.ej fin de E/S)
  - Ejecución de un proceso *batch* (podría comenzar "listo" o "suspendido y listo")

Colas de procesos

- El S.O. organiza los PCB en colas de espera por el procesador o por los dispositivos de E/S. (colas de planificación: cola de procesos, colas de dispositivos)

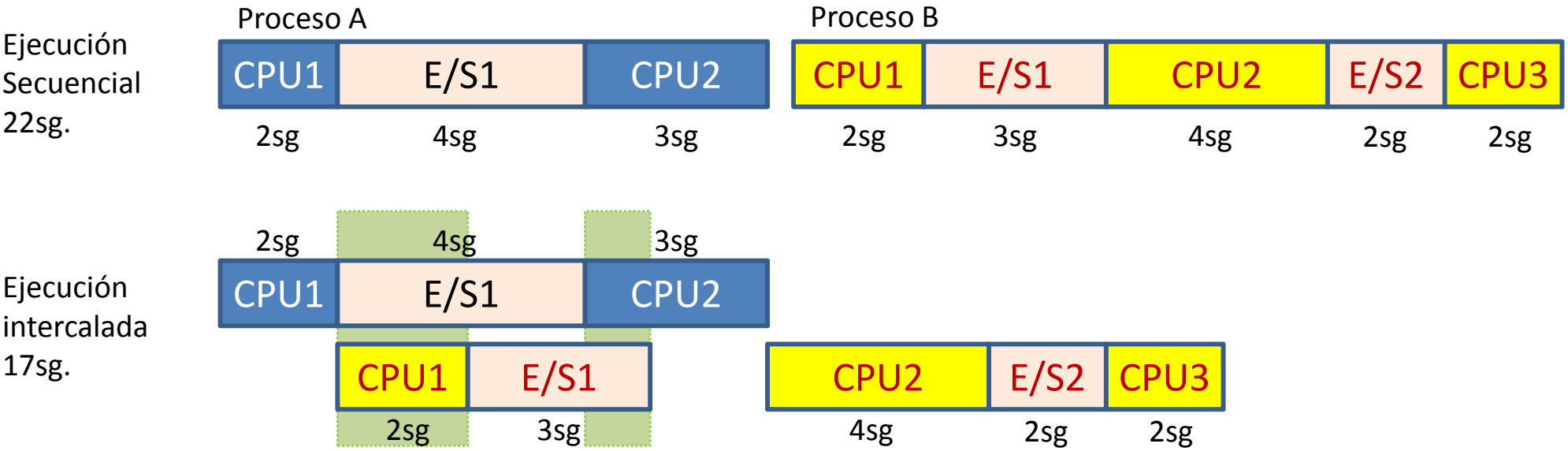


Multiprogramación y Planificación de la CPU

Fariña, Pedreira: LBD@2010

- Todo proceso típicamente alterna ráfagas de CPU y ráfagas de E/S.

- Multiprogramación: Mientras se espere por E/S, otro proceso puede usar la CPU. Mejora el uso de la CPU.



- **Planificación**

  OBJETIVO: Reparto del tiempo de CPU entre los procesos.

Planificación de la CPU

- Planificador. Objetivos:
  - Reparto justo (tiempos de espera de cada proceso?)
  - Maximice uso de CPU
  - Maximice número de usuarios interactivos
  - Debe ser predecible
  - Minimización de sobrecarga.
    - Obtener cuál es el siguiente (tiempo de decisión)
    - Cambios de contexto (salvar estado BCP1/ restaurar estado BCP2)
  - Equilibrio de uso de recursos
  - Manejará Prioridades de procesos.

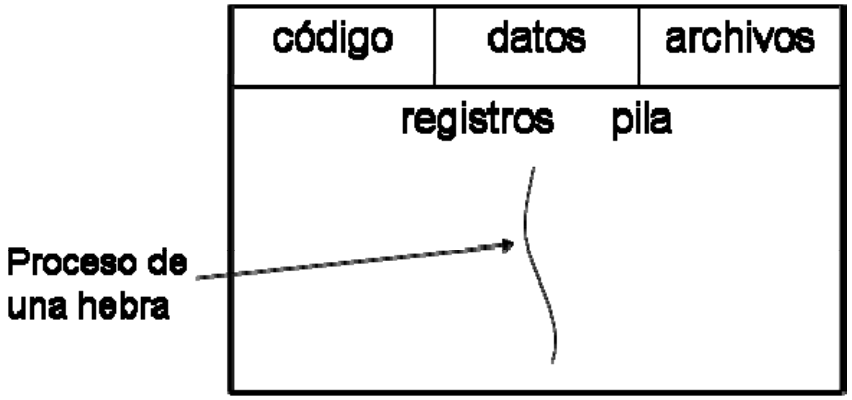
- Múltiples algoritmos de planificación
  - FIFO, Shortest Job First, Shortest Remaining Time First, Round Robin, etc.

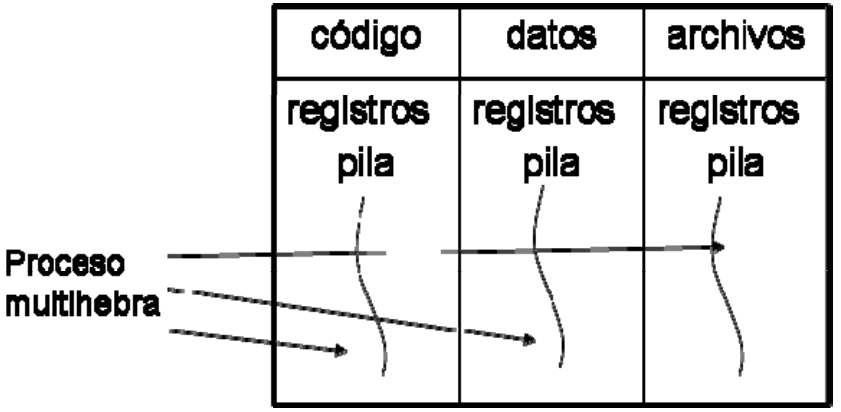
Procesos ligeros, threads, hilos, hebras

- Procesos ***monothread***, cuentan con un único thread o hilo de ejecución.
  - Proceso contiene:
    - Recursos: Mapa de memoria + BCP + ficheros abiertos +, …
    - Flujo de ejecución: instrucciones que va a ejecutar el proceso. → Hilo de ejecución o *thread*.
- Proceso ***multithread***
  - Tiene recursos (que se comparten entre los threads)
  - Cada thread tiene su propio flujo de ejecución, y las informaciones requeridas para ello:
    - Estado procesador (PC y registros)
    - Pila
    - Estado (ejecutando, listo o bloqueado)
    - Prioridad de ejecución
    - Bloque Control del Thread.
  - Permite conseguir concurrencia dentro del proceso, pues dichos threads se pueden ejecutar dentro de un mismo proceso.
    - Ejemplo: impresión y botón para cancelar impresión.
    - Servidor web (multithread para poder recibir numerosas peticiones).

Procesos ligeros, threads, hilos, hebras

- Un proceso que tiene varios hilos puede realizar varias tareas a la vez. La hebra comprende:
  - Identificador de hebra
  - Contador de programa
  - Conjunto de registros
  - Pila,
  - BCT (Bloque control Thread).
- Comparte con las otras hebras que pertenecen al mismo proceso:
  - Sección de código
  - Sección de datos
  - Otros posibles recursos: archivos abiertos, señales, contabilidad,…





Procesos ligeros, threads, hilos, hebras

- Ventajas vs Procesos.
  - **Capacidad de respuesta**: si un proceso se bloquea por alguna causa, el proceso puede continuar con otra parte del código. ¿?
  - **Compartición de recursos**: puede haber varios threads accediendo a los mismos recursos (memoria, ficheros abiertos)
  - **Economía**: el coste de crear procesos es muy superior al de crear hebras. Los cambios de contexto también son más latosos en procesos
  - **Utilización de arquitecturas multiprocesador**: la arquitectura multiprocesador puede ayudar en este sentido. Un proceso monohebra sólo se puede ejecutar sobre un procesador.
    - Multihebra a nivel de usuario (librerías). SO desconoce la existencia de threads: Capa de software para crear, planificar (depende de la aplicación), activar y terminar threads.
    - Multihebra a nivel del kernel *(Kernel* del S.O.*)*. El núcleo del SO crea, **planifica**, activa y termina threads.

Bloque II – Sistemas Operativos

4. Introducción a los Sistemas de Ficheros

Ficheros

- Def. fichero.
  - Un fichero es una unidad de *almacenamiento* ***lógico*** no volátil que agrupa un conjunto de información relacionada entre sí bajo un mismo ***nombre***.
  - Un fichero es una *abstracción* que bajo el nombre de fichero oculta las operaciones de bajo nivel para operar con datos e información almacenados en un dispositivo de almacenamiento secundario (como una cinta magnética, un disco,…)

Ficheros

- Pdv. Del usuario.
  - Contenedor de información: datos y programas.
    - Agrupados en directorios.
  - Ficheros especiales que modelan la E/S como un fichero más. Permiten acceso a:
    - Dispositivos de carácter (terminales, impresoras,…)
    - Dispositivos de bloque (cintas, discos,…)
- Pdv. Del S.O.
  - Forma de estructurar el almacenamiento 2ario.
  - Sistema Ficheros define cómo: estructurar ficheros, identificarlos, implementación, acceso, protección, etc.
  - Un fichero tiene **atributos**:
    - Nombre (nivel usuario) e identificador único (id = número interno)
    - Tipo de fichero
    - Mapa del fichero (localizar dispositivo y bloques con su info).
    - Protección
    - Tamaño (bytes, nº bloques,…)
    - Información temporal (creación, último acceso,…)
    - Información de control (oculto, de sistema, normal, directorio,…)

Ficheros

- El S.O. debe dar:
  - una **estructura** de fichero **genérico:**
    - Que soporte todos los tipos de ficheros existentes,
    - un mecanismo de nombrado,
    - facilidades de protección, y
  - servicios para explotar el almacenamiento 2ario y el sistema de E/S.

- Descripción de un fichero (atributos) se almacena en:
  - i-nodo: Unix
  - registro de MFT (Master File Table): NT/NTFS windows.
  - Entrada de directorio: FAT (File Allocation Table): MSDOS.

Bloques y fragmentación interna

- Estructura disco:
  - Cilindro > cara > sector. El disco trabaja a nivel de sector (p.ej 512bytes).

- El SO. Divide el disco en **bloques** de 1,2,4,8… sectores.
  - El S.O. usa sectores lógicos. Están numerados (1,2,3,4,5,…)
  - No maneja (cilindros, caras ni sectores) → lo hace el controlador de disco.
    El controlador hace la conversión.
  - El bloque es la unidad de E/S del S.O.
  - Tamaño de bloque del S.F. es generalmente “configurable” (dep S.F.)
    - Ej: mkfs.ext4 –b 4096

- Los dispositivos de almacenamiento guardan “bloques”, no “ficheros”.
- Unix/windows. **Estructura lógica de un fichero** = secuencia de bytes.
  - Es posible acceder a cualquier byte del fichero si se conoce su desplazamiento desde el origen del fichero.

Bloques y fragmentación interna

- Todo fichero requiere al menos 1 bloque.
  - Un bloque no puede ser compartido con datos de 2 ficheros distintos.
  - **Ejemplo**: tamaño de bloque = 4kb
    - Cada fichero de mi S.F. necesita al menos 1 bloque.
    - 1 fichero de 2 bytes ➔ 1 bloque
    - 1 fichero de 6328 bytes ➔2 bloques.

- **Fragmentación interna**.
  - La asignación de espacio “por bloques” hace que se desperdicie espacio.
  - A mayor tamaño de bloque
    - Mayor espacio desaprovechado.
    - Acceso más rápido (hay que leer menos bloques)
  - **Ejercicio**: Un fichero de 13310 bytes tiene asignados 4 bloques de 4Kb. ¿en qué bloque está el byte nº 12000 de dicho fichero?¿Cuántos bytes se desperdician debido a la fragmentación interna?¿Cuántos bytes se desperdiciarían si el tamaño de bloque fuese de 1kb?

Ficheros: Métodos de acceso

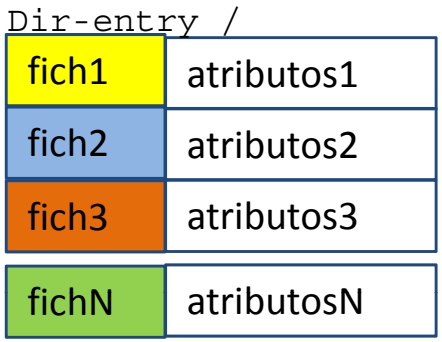
- **Acceso secuencial**.
  - Los bytes del fichero se leen en orden.
    - Para leer una posición es necesario haber leído todas las anteriores.
  - Operaciones: leer siguiente / escribir siguiente.
  - Interesante: Acceso eficiente cuando se leen todos los datos de un fichero desde el principio al final de forma incremental (p.ej. Editor de textos).
  - Dispositivos: Cinta (acceso secuencial), pero también en discos (que permiten acceso directo a cualquier bloque).

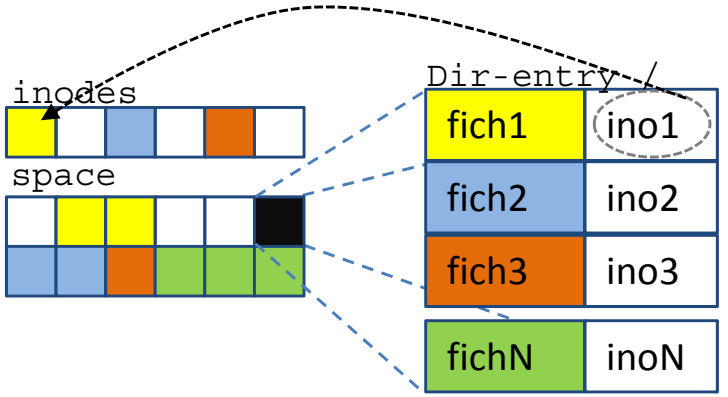
- **Acceso directo o aleatorio**.
  - Es posible determinar qué bloques constituyen el fichero.
    - Es posible acceder a cualquier bloque (y byte dentro del bloque) sin haber procesado los anteriores
  - Operaciones: read/write (leer/escribir siguiente) y seek(posición).
  - Acceso secuencial sigue siendo posible.
  - Dispositivos: Han de permitir direccionar cualquier bloque. Ej: Disco.

Directorios

- Un directorio es un objeto que relaciona de forma unívoca el **nombre** de usuario de un fichero con el **descriptor interno** usado por el S.O.
  - Permiten organizar y proporcionar información acerca de la estructuración de los ficheros en los S.F.
- Un directorio contiene tantas "entradas" como ficheros hay accesibles en él.
  - Cada entrada relaciona un nombre de fichero con su descriptor interno (inode, registro MTF,…).
- Visión lógica: oculta los detalles de cómo está implementado.
  - Estructura lineal, árbol ordenado, función hash,…
    - Estructura lineal es típica: Unix.

Directorios: estructura lineal



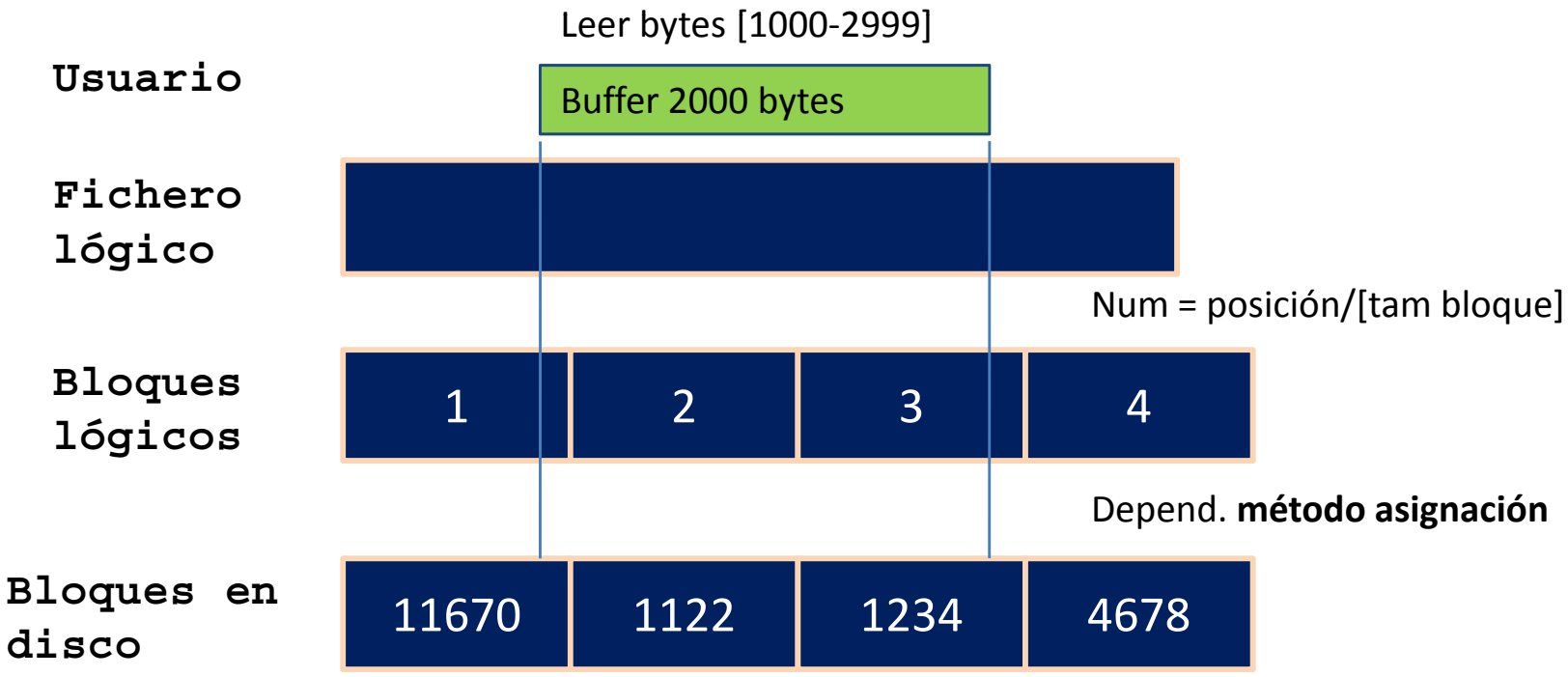


Ino-i = Id acceso a estructura con atributos del fichero

- Crear fichero:
  - Asignar espacio a fichero y **añadir entrada directorio**.
- Borrar fichero:
  - Desasignar espacio y marcar entrada directorio como “libre”.
- Leer/escribir fichero.
  - A partir de la entrada de directorio correspondiente podemos acceder a la información sobre el fichero (ej. i-node), y ver dónde están sus bloques de datos.
  - A continuación se leen dichos bloques.

Offset → bloque lógico → bloque disco

- ¿Cómo traducir de posición (byte) de un fichero a bloques lógicos y después a bloques de disco?



Espacio libre??
Asignación bloques??

Métodos de asignación

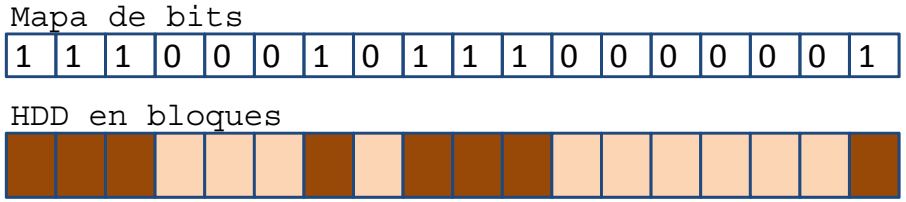
- Todo S.F. es considerado una serie de bloques.
  - Cuáles están libres?
  - Qué bloques pertenecen a qué fichero?

S.O.
Contabilidad!!

- Contabilidad:
  - Ha de residir en el propio S.F. (me puedo llevar el HDD a otro equipo!!)
- Gestión de espacio libre
  - Mapa de bits
  - Lista enlazada
  - Lista enlazada de bloques de índices
  - …

Métodos de asignación: Espacio libre

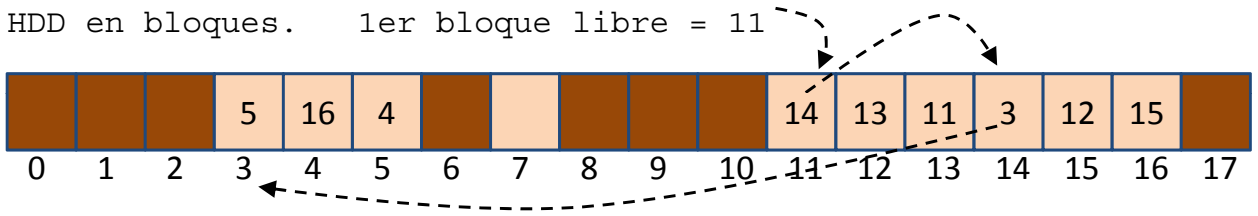
- **Mapas de bits**.
  - Utiliza tantos bits como bloques tiene el S.F.
  - 1 = bloque ocupado, 0 = bloque libre.



  - Simple y eficiente (si disco no muy lleno y/o fragmentado), y si el mapa de bits reside en memoria.
  - Ocupa mucho si dispositivo muy grande. (numBloques/32 bits)
    - Ocupa siempre lo mismo indep de si hay mucho o poco espacio libre.
  - **Ejemplo**: Tengo un S.F. de 4Gb, el tamaño de bloque es 4kb. Se utiliza un mapa de bits para gestionar el espacio libre ¿Cuánto espacio (en bytes) estoy usando?¿Cuántos bloques de ese disco se utilizan para almacenar dicha información?

Métodos de asignación: Espacio libre

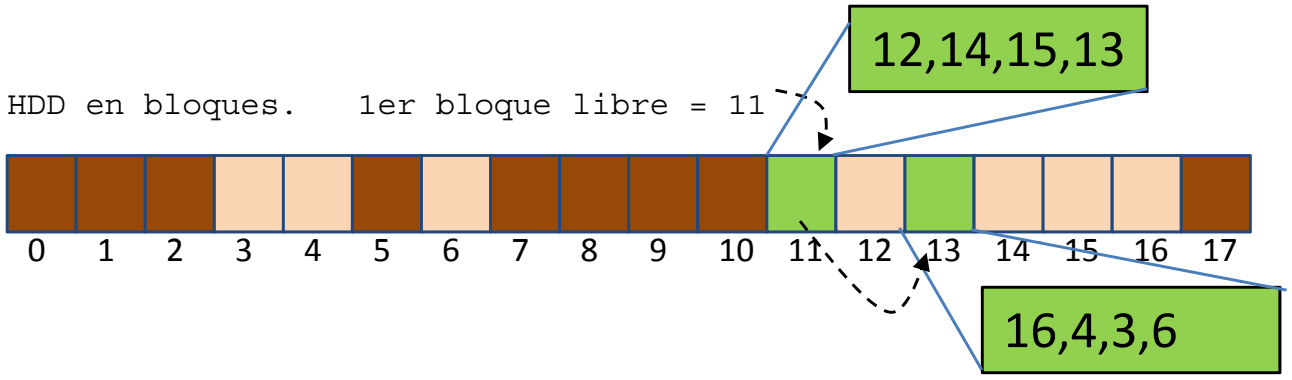
- **Lista enlazada de bloques libres**.
  - Se crea una cadena de bloques libres.
    - En cada bloque libre se almacena cuál es el siguiente bloque libre.
    - La posición del primer bloque libre se guarda en memoria.
  - Recorrer la lista de libres requiere 1 E/S (leer 1 bloque).
    - Lento salvo si disco muy lleno o fragmentado.
  - No indica si el espacio libre está contiguo o no.



Métodos de asignación: Espacio libre

- **<u>Lista enlazada de bloques de índices</u>** (agrupaciones de B. libres)
  - Similar al anterior, pero evita el problema de E/S.
    - En lugar de que cada bloque libre contenga sólo la dirección del siguiente bloque libre.
    - Se utiliza todo el bloque para almacenar N números de bloques libres.
    - Si T = tamaño bloque (bytes)

      → cada bloque contiene T/4 núms_de_bloques_libres (asumiendo que se usan 32bits por entero)



Métodos de asignación: bloques de datos

- Métodos de **asignación contigua**
  - Cada fichero ocupa un conjunto de bloques consecutivos en disco.
  - Muy rápido: Acceso directo.
    - accesos a 1er bloque y resto bloques consecutivos.
  - Acceso directo a cualquier bloque: Calculando a partir del 1er bloque
  - **Fragmentación externa.**
    - Existen huecos "pequeños" que no se pueden usar entre bloques asignados a 2 ficheros diferentes
  - Problemas:
    - hacer crecer un fichero → ¿hay hueco contiguo?
    - Reducir tamaño fichero → surgen huecos: fragmentación.
    - Desfragmentación permitirá evitar "fragmentación externa", pero lleva mucho tiempo!!

Métodos de asignación: bloques de datos

- Métodos de **asignación discontinua**
  - Para cada fichero, se le asigna el primer bloque libre.
  - Fichero puede crecer mientras exista espacio libre (o límite de tamaño impuesto por el S.F.)
  - **No** se genera **fragmentación externa**.
  - Problemas: Los bloques de cada fichero pueden estar dispersos
    - Debemos conocer cuáles son, y
    - En qué orden debemos recorrerlos.
    - En principio, no soporta acceso "directo" a cualquier *offset*.
      - P.ej en asignación enlazada → ir recorriéndolos
      - Pero técnicas "indexadas" soportan acceso rápido a cualquier posición (y directo a algunos offsets.
  - Tipos:
    - Asignación enlazada,
    - Asignación mediante índices,
    - Árboles balanceados,…

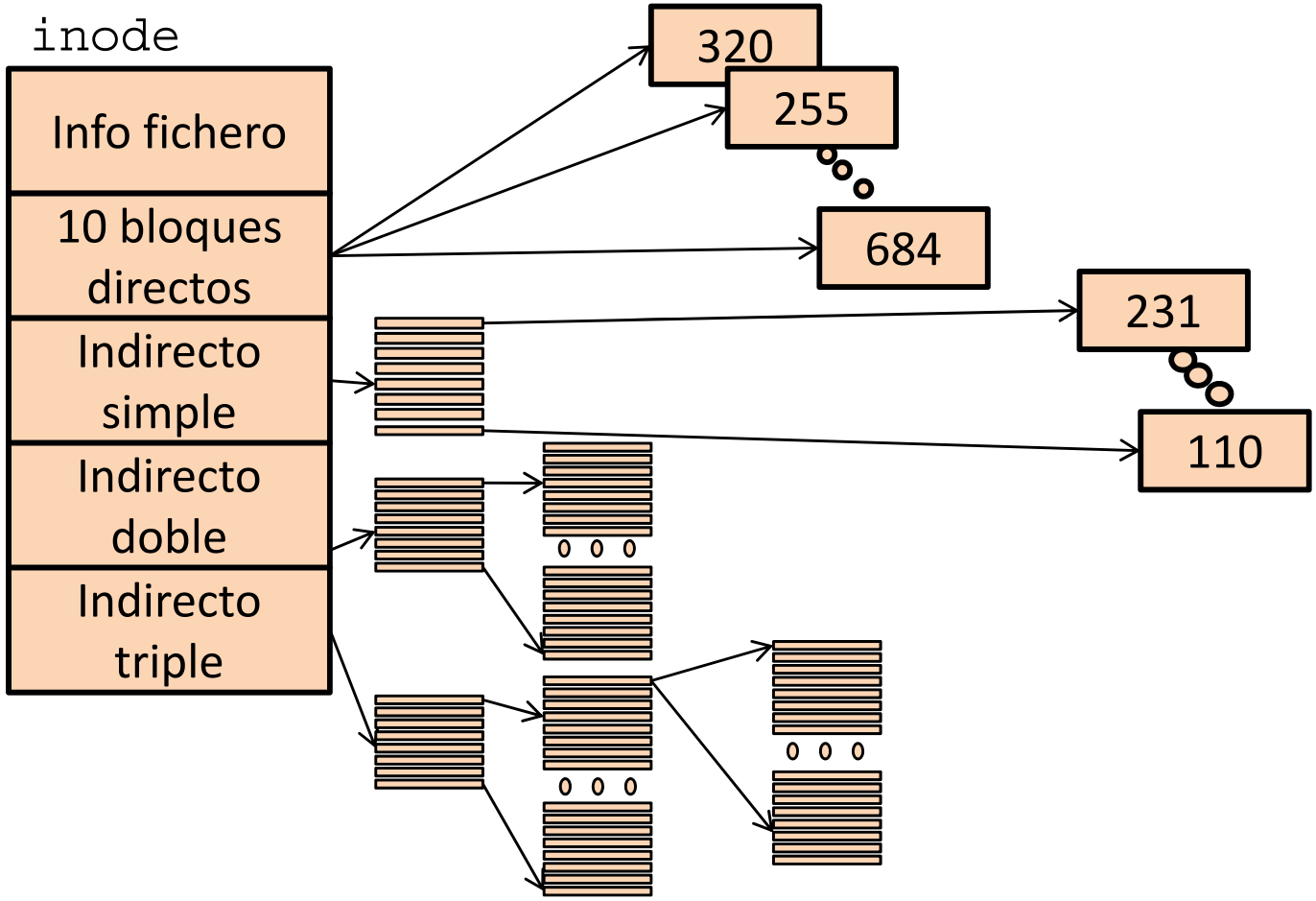
Métodos de asignación: bloques de datos

- Métodos de **asignación discontinua**
  - **Asignación enlazada**
    - Para cada fichero se anota dónde está el primer bloque.
    - Cada bloque contiene un apuntador (p.ej un entero de 32/64bits) que indica cuál es el siguiente bloque.
      - No permite acceso aleatorio, y complica el cálculo de en qué bloque está determinado offset del fichero.
      - Hay que leer el siguiente bloque para saber por dónde continuar.
  - **Asignación mediante índices** (Unix/linux: usando inodes)
    - En lugar de guardar un apuntador en cada bloque de datos (mezclando apuntador y datos)…

      …Se utilizan algunos bloques para "apuntar" qué bloques contienen los datos de cada fichero.

Métodos de asignación: índices

- Ejemplo



- Si tamaño bloque = 4kb, y un bloque se identifica con un número de 32bits, ¿cómo llegar al bloque de datos que contendrá el byte con offset 49152.
- ¿Cuál es el tamaño máximo que puede tener un fichero en este S.F. si no queremos usar apuntadores indirectos triples?

Ejemplos: Unix System V

- Boot: arranque
- Superblock: metainformación: tam bloque, tamaño nodo-i.
- Gestión espacio libre: mapa bits.
  - Ej. Cuánto ocupa mapa de bits en un S.F. de 64Gb y tam bloque = 2Kb
- Inodos (representación física fichero). Uso espacio de las “lista de inodos”.
  - Info (entre otros): propietario, grupo, tamaño (libre:¿-1?), fechas (creación,modificación,acceso).

    Punteros directos (10), ptr indirecto simple/doble/triple
- Bloques de datos.
  - Asignación de espacio → mirando en mapa de bits.

Ejemplos: Unix System V

Dir raíz

.	
..	
bin	
boot	3
dev	
etc	
home	

Al buscar "boot" obtengo el inode 3

El inode 3 es del "boot"

Modo, tamaño, tiempos
99
isDir

El inode3 indica que "boot" está en el bloque 99

El bloque 99 contiene las entradas del "/boot"

.	
..	
abi	
grub	25
vmlinuz	
etc	
...	

Al buscar "grub" obtengo el inode 25

El inode 25 es del "grub"

Modo, tamaño, tiempos
68
isDir

El inode 25 indica que "/boot/grub" está en el bloque 68

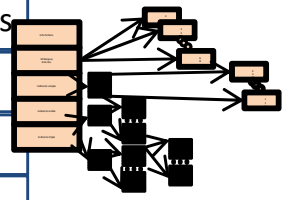
El bloque 99 contiene las entradas del "/boot/grub"

.	
..	
default	
devicemp	
Menu.lst	443
etc	
...	

Al buscar "menu.lst" obtengo el inode 443

El inode 443 es del "menu.lst"

Modo, tamaño, tiempos
→
isFile

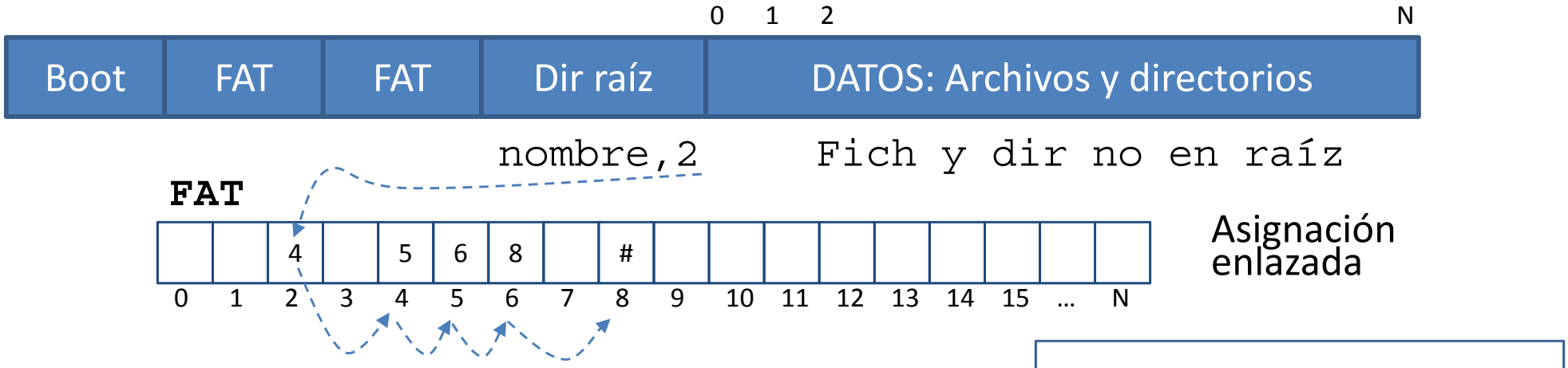


El inode 443 apunta a bloques_directos, indirectos,… con los datos del fichero menu.lst

- Ejemplo: pasos para buscar /boot/grub/menu.lst

Ejemplos: msdos-FAT (File Allocation Table)

Fariña, Pedreira: LBD@2010



Nombre|extensión|Atribs|[vacío]|hora|fecha|nº 1er bloque|Tamaño

Entrada directorio

- 1 entrada en la FAT por cada bloque.
  - 0 → libre
  - >0 → indica siguiente bloque asignado
  - # → último bloque asignado.
- Asignación de espacio: buscar en la FAT
- FAT16, FAT32,… cada entrada de FAT = 16/32 bits → tamaño máximo de fichero varía, nº ficheros máximo varía.
  - Ej: FAT16 cuenta bloques de tamaño 2kb,4kb,…,32kb → 2^16*2^11=128MB, 2^16*2^15=2GB
  - Ej: FAT32 cuenta bloques de tamaño 512bytes → 2^9 * 2^32 = 2TBytes
- Ej: S.F. de 64Gb. Tamaño bloque = 4kb. ¿Cuánto ocupa la FAT?

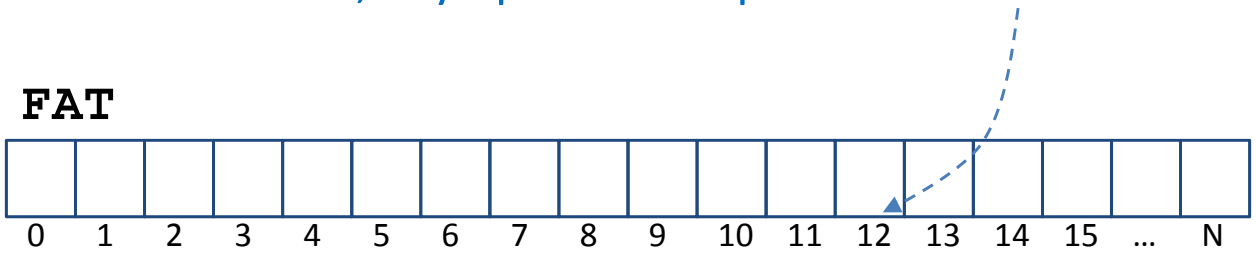
  16MegaBloques x 4 bytes/bloque → 64Mbytes FAT

Ejercicios

1. Sea un fichero “A” de 10.000 bytes que se almacena en un Sistema de ficheros (SF) cuyo tamaño de bloque son 2kb.
   - ¿Cuántos bloques son necesarios para almacenar sus datos?
   - ¿Cuánto espacio se desaprovecha?
   - Si los bloques asociados a cada fichero se numeran como: 0,1,2,3,4… ¿En qué bloque de dicho fichero se almacenan los bytes que van desde el 8100 al 8120?
2. Asumiendo que en el SF anterior se realiza asignación contigua de espacio, y que el primer bloque del fichero “A” es el 65. ¿Cuáles son los bloques del disco que son utilizados por el fichero “A”?
3. Sea un fichero “B” de 8.192 bytes que se almacena en un Sistema de ficheros (SF) cuyo tamaño de bloque son 2kb. ¿Cuántos bloques son necesarios para almacenar sus datos si se realiza asignación contigua?¿Y si se realiza asignación enlazada?
4. En un SF con inodos, donde se guardan (para cada fichero) 10 ptrs a bloques directos y 2 punteros indirecto simple, 1 puntero indirecto doble y 1 puntero indirecto triple: ¿cuál es el tamaño máximo que puede tener un fichero? (asúmase que se usan bloques de 4Kb, y que cada bloque se puede referenciar con un entero de 32bits = 4bytes).

Ejercicios

5. Tenemos un sistema FAT16, cuyo tamaño de bloque es de 2KB, y en él un fichero "C" de 2KB de tamaño, cuyo primer bloque es el nº 12.



Tras abrir "C" con el pico, y haberlo modificado durante 2 horas, queremos guardar los cambios realizados. Esto hace que el fichero aumente su tamaño en 11000 bytes. Si en el instante actual los bloques libres existentes en el SF son los siguientes: el 4,5,9,10,14,1,15,2 y 3 (en ese orden). Modifíquese la FAT para asignarle a "C" los bloques necesarios para que almacene sus 2048 +11000 bytes.

6. Tenemos un SF de tipo FAT16 con tamaño de bloque = 16KB. Si el tamaño medio de un fichero es de 7000 y tenemos 1000 ficheros en dicho SF actualmente
   - ¿Cuánto espacio estamos desperdiciando debido a la fragmentación interna?
   - ¿Y a la externa?