# El sistema de ficheros.

Sistemas Operativos
Tema 5.



## Introducción

- Para administrar los dispositivos de almacenamiento masivo (cintas, discos), es necesario que el SO implante el concepto abstracto de fichero.
- Los ficheros almacenan datos y programas y para facilitar su uso se organizan en estructuras denominadas directorios.
- Es necesario controlar “quien y como” puede acceder a un fichero.
- Para esto se utilizan ciertas técnicas de protección y seguridad.

## Introducción

- Para almacenar sistemas de ficheros se utilizan medios físicos como: cintas, discos magnéticos.
- El S.O. a partir de las características del sistema físico definirá una unidad lógica de almacenamiento.

Así un **sistema de ficheros** vendrá dado por:

- un conjunto de ficheros.
- estructura de directorios.

¿Como definir estos conceptos?



## Concepto de fichero.

# ¿Que es un fichero?

- Ente que sirve para el almacenamiento de información.
- Conjunto de información relacionada definida por su creador.
- En UNIX la abstracción de fichero consiste en una secuencia ordenada de bytes residentes en memoria secundaria.
- En general es una secuencia de bits, bytes, líneas o registros definidos por su creador y el usuario.

## Concepto de directorio

### Directorio como tipo especial de fichero

- Es un mecanismo o estructura para organizar los ficheros.
- Con esta estructura, si se sobrepasa la capacidad física del dispositivo se pueden usar otras unidades de almacenamiento incluso en computadoras diferentes.
- Algunos sistemas utilizan dos estructuras diferentes:
  - directorio del dispositivo: se almacena en cada dispositivo e informa de las propiedades físicas de los ficheros, ubicación física.
  - directorio de ficheros: describe las propiedades lógicas de los ficheros, nombre, tipo...



## Fichero como entrada en un directorio

- La entrada en un Directorio es diferente según el sistema operativo que usemos.
- En general contiene la siguiente información:

| | |
|---|---|
| **Nombre de fichero** | nombre simbólico dado por el usuario. |
| **Tipo de fichero** | para sistemas que permiten varios tipos. |
| **Ubicación** | puntero al dispositivo y su posición en él. |
| **Tamaño** | tamaño del fichero en bits,bytes, bloques.. etc. |
| **Posición actual** | puntero a la posición actual en el fichero si se está realizando una operación de lectura o escritura. |
| **Protección** | información sobre el control de acceso. |
| **Recuento de uso** | valor que indica el número de procesos que están usando el fichero (que lo han abierto). |
| **Propietario** | usuario que creo el fichero. |
| **Hora, fecha…** | información referente a hora y fecha de creación, ultima modificación, último acceso. |

## Estructura de Directorio

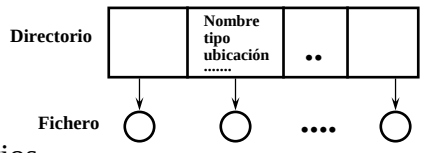
- En general un directorio es una estructura de datos que contiene un conjunto de entradas o registros que hacen referencia a un fichero u otro directorio.
- En UNIX los directorios se implementan como ficheros.
- En otros sistemas operativos se implementan como estructuras de datos sobre el disco.
- Operaciones: búsqueda , creación y eliminación de ficheros, listado del directorio... etc.



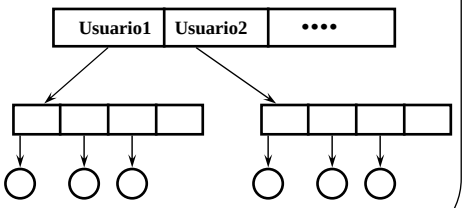


## Estructura plana

- Un solo nivel:
  - Directorios de un solo nivel
  - Ficheros con nombres únicos
  - No hay distinción entre usuarios

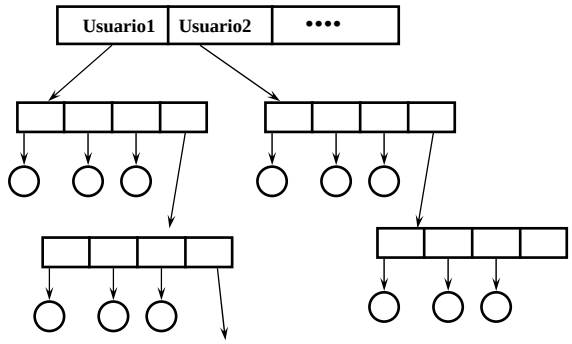



- Dos niveles:
  - Directorios maestro con directorios de usuarios
  - Usuarios aislados



## Estructura de Directorio

**Estructura jerárquica o de árbol de altura arbitraria**



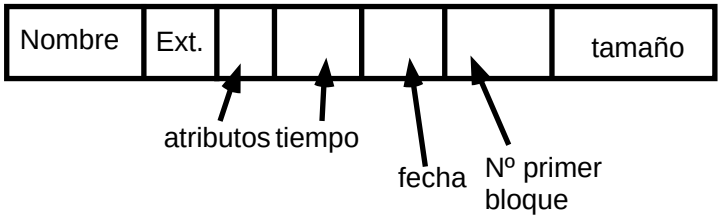



## Ejemplo de directorio en MS-DOS

**MS-DOS**

- Estructura arborescente
- Una entrada en un directorio proporciona un puntero al primer bloque de datos (FAT)
- Existe un directorio raíz de tamaño fijo.

**Entrada en un directorio**



## Ejemplo de directorio en UNIX

# UNIX/MINIX

- Estructura arborescente
- Nº arbitrario de entradas
- 2 entradas predeterminadas "." y ".."

**Entrada en un directorio**

| Nº nodo-i | Nombre del fichero |
|---|---|



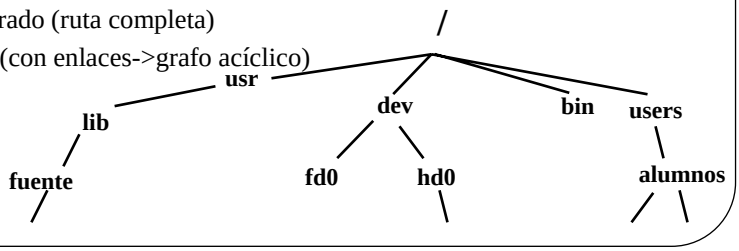
## Ejemplo de directorio en UNIX

# UNIX/MINIX: operaciones sobre estructura de árbol

- Acceso a un fichero dando su "ruta o path" absoluta o relativa.
  - absoluta: comienza en el raíz y se da la ruta hasta el fichero.
  - relativa:se define la ruta desde el directorio actual.
- Cambio de directorio: mediante la llamada al sistema chdir(dir) Toma el directorio que se especifica como el actual.
- Borrado de directorios:
  - si está vacío: se elimina la entrada del directorio que lo contiene.
  - si no vacío: los S.O. borran primero su contenido y después el directorio.

## Directorio como grafo acíclico: enlaces

- Dos o más usuarios puede compartir ficheros o directorios como si todos fuesen propietarios.
- Con un fichero compartido solo hay un fichero, modificaciones de un usuario son vistas instantáneamente por el otro.
- Estructura jerárquica
  - Nombrado (ruta completa)
  - Árbol (con enlaces->grafo acíclico)

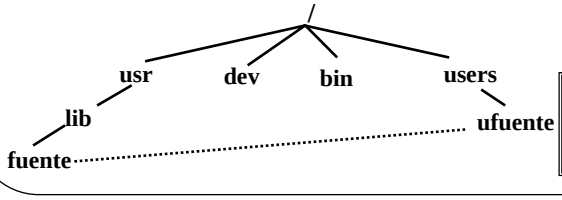







## ¿Como compartir ficheros o directorios?

- **Duplicando la información.**
  - Así se obtienen entradas idénticas. (cp)
- **Creando enlaces.**
  - Un enlace es un puntero a otro fichero o directorio. En la entrada al directorio del fichero o directorio compartido se indica que hay un enlace y el nombre real del fichero o directorio. En UNIX se crean enlace sobre ficheros con la orden "ln".




**$cd /users**
**$ ln /usr/lib/fuente ufuente**

**Precaución:**
Hay que evitar la creación de **ciclos** al crear nuevos enlaces.

## Directorio (Unix) como grafo acíclico: enlaces

# ¿Como eliminar ficheros compartidos?

- **Si se usaron enlaces simbólicos**
  - Al eliminar un enlace no se modifica el fichero original
  - Al eliminar la entrada del fichero original puede quedar enlaces sueltos

**!! El sistema debe tener algún control sobre los enlaces!!**

- En UNIX el S.O. Lleva una cuenta de los enlaces que se realizan
  - Si se borran/crean enlaces se reduce/aumenta la cuenta.
  - Cuando la cuenta es cero se puede borrar el fichero porque ya no hay ninguna referencia a el.



## Operaciones sobre Ficheros y Directorios

# Operaciones

- El S.O nos ofrece las “llamadas al sistema” para la gestión y manejo de ficheros y directorios.

### Llamadas al Sistema de UNIX

**Gestión de ficheros**

```
fd = creat(n,md)
fd = mknod(n,md,dir)
s = close(fd)
n = read(fd, buff,nb)
n = write(fd, buff,nb)
pos = lseek(fd, pos, rel)
s = stat(n, &buff)
s = fstat(fd, &buff)
fd = dup(fd1)
s = pipe(&fd[0])
s = ioctl(fd, pet, arg)
```

**Gestión de Directorios**

```
s = link(n1, n2)
s = unlink(n)
s = mount(sp, n, rwbits)
s = umount(esp)
s = sync()
s = chdir(ndir)
s = chrot(ndir)
```

**Protección**

```
s = chmod(n, md)
uid = getuid()
gid = getid()
s = setuid()
s = setgid()
s = chown(,n prop, gr)
mant = umask(m)
```

## Operaciones sobre Ficheros y Directorios

**!!Todas las operaciones implican la búsqueda de la entrada en el directorio!!**

**TABLA DE DESCRIPTORES DE FICHERO (abiertos):**

- Al abrir un fichero se copia su entrada en esta tabla, asociándola a un índice (entero).
- Al realizar una operación sobre el fichero se utiliza ese índice para referirse a él.
- Al cerrar el fichero su entrada se borra de esta tabla.



## Operaciones sobre Ficheros y Directorios

# Algunas Operaciones sobre Ficheros

- **Creación.- Pasos:**
  - **asignación de espacio libre en el sistema de ficheros.**
  - **anotar el nuevo fichero en el directorio (nueva entrada).**
- **Escritura.- A la llamada al sistema se le pasa como argumentos el nombre del fichero y la información a escribir.**
  - **el sistema localiza la ubicación en el directorio de ese nombre.**
  - **usa el puntero a la posición actual para saber donde escribir actualizando la posición del puntero después de la operación.**
- **Lectura.- A la llamada al sistema se le pasa el nombre del fichero.**
  - **el sistema localiza la ubicación en el directorio de ese nombre.**
  - **pone un puntero al bloque que leerá, lo lee y actualiza la posición del puntero.**

NOTA: *Normalmente los ficheros tienen un solo puntero para lectura y escritura cuya posición se modifica al realizar dichas operaciones.*

## Algunas Operaciones sobre Ficheros

- **Posicionamiento.- (lseek en UNIX). Modificar la posición del puntero de un fichero que se ha abierto previamente.**
  - **El posicionamiento es relativo a una posición que se le pasa a la llamada al sistema.**
- **Eliminación.- Pasos:**
  - **Localizar entrada en el directorio por el nombre del fichero.**
  - **Liberar el espacio que ocupa e invalidar su entrada en el directorio.**



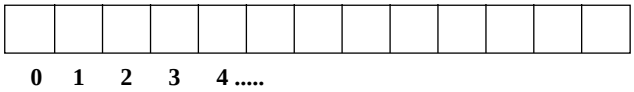


## Métodos de almacenamiento de ficheros

### ¿Como almacenar un fichero en un dispositivo?

**Un fichero se almacena en:**
- **posiciones consecutivas**
- **lista enlazada (MS-DOS)**
- **indexado (UNIX)**

**Ficheros implementados como una secuencia ordenada de bloques**



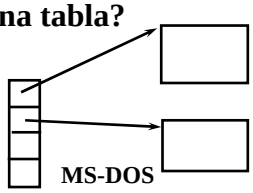
## Lista enlazada

- El fichero se almacena en disco en bloques (no necesariamente consecutivos).
- Se genera una lista enlazada con los bloques que contienen los números de los bloques que hay ocupados por el fichero.
- La lista estará enlazada por punteros.

**¿donde se ubica el enlace?**

**1. ¿en el bloque?**


**2. ¿En una tabla?**







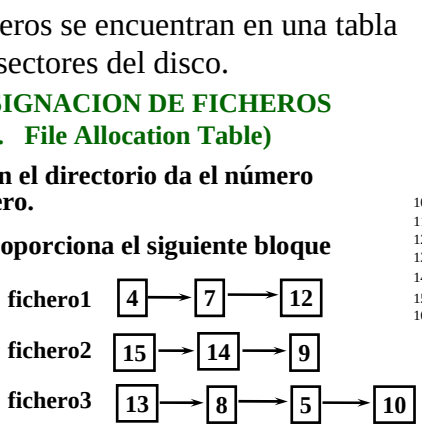

## Lista enlazada de bloques (MS-DOS): FAT

- Los punteros de los ficheros se encuentran en una tabla situada en los primeros sectores del disco.

**TABLA DE ASIGNACION DE FICHEROS**
**(F.A.T. File Allocation Table)**

- **La entrada de un fichero en el directorio da el número del primer bloque del fichero.**
- **Esta posición en la FAT proporciona el siguiente bloque y así sucesivamente.**

**XX = tamaño del disco**
**EOF = fin fichero**
**FREE = bloque libre**
**BAD = bloque defectuoso**




| | |
|---|---|
| 0 | XX |
| 1 | XX |
| 2 | EOF |
| 3 | FREE |
| 4 | 7 |
| 5 | 10 |
| 6 | BAD |
| 7 | 12 |
| 8 | 5 |
| 9 | EOF |
| 10 | EOF |
| 11 | BAD |
| 12 | EOF |
| 13 | 8 |
| 14 | 9 |
| 15 | 14 |
| 16 | FREE |
| | FREE |
| | FREE |
| | FREE |
| | FREE |

## Métodos de almacenamiento de ficheros

### MS-DOS

**Los punteros se encuentran en una tabla aparte situada en los primeros sectores del disco: Tabla de asignación de ficheros (FAT- file allocation table)**

| Disco tamaño Punteros | | 320K 12 | 360K 12 | 64M 16 |
|---|---|---|---|---|
| Tamaño de la FAT | posiciones | 320 | 360 | 64K |
| | bytes | 480b | 540b | 128K |
| | sectores | 1 | 2 | 256 |

**320 posiciones x 12 bits = 480 bytes < 512**
**360 posiciones x 12bits = 540 bytes > 512**
**64K pos. x 16 bits = 128K bytes -> 256**

- **¿Se guarda la FAT en memoria?**
- **Aleatoriedad de la organización**



## Almacenamiento indexado

### Almacenamiento indexado

- Problema de la FAT: punteros mezclados aleatoriamente, puede ser necesario mantener toda la FAT en memoria para tener abierto un solo fichero.
- Un nuevo método: listas de bloques de los ficheros en sitios distintos.

### UNIX

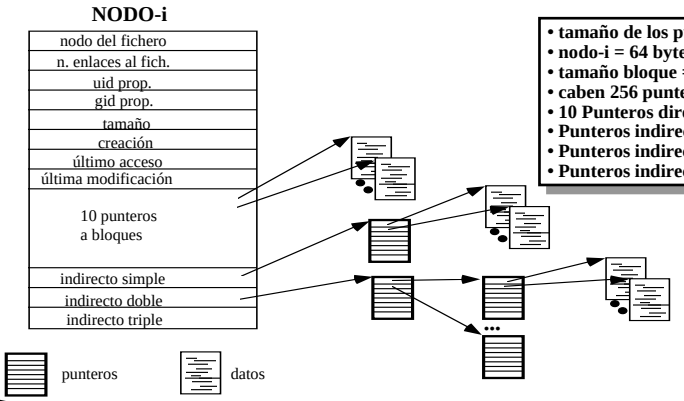
- Existe una tabla asociada (en disco) llamada nodo-i que contiene la información referente a un fichero. Esta tabla es una estructura definida en C, que está almacenada en disco.

## Almacenamiento indexado

### Ficheros en UNIX

- **Estructura de datos de cada fichero independiente del resto**
- **Estructura llamada: nodo-i**

**UNIX**




- **tamaño de los punteros: 32 bits**
- **nodo-i = 64 bytes**
- **tamaño bloque = 1024 bytes**
- **caben 256 punteros en un bloque**
- **10 Punteros directos**
- **Punteros indirectos simples**
- **Punteros indirectos dobles**
- **Punteros indirectos triples**



## Almacenamiento indexado

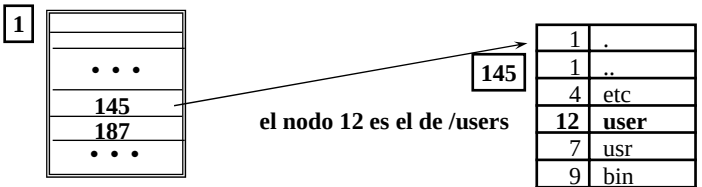
### Capacidad de direccionamiento

| | |
|---|---|
| **p. simples** | **10K** |
| **p. indirectos** | **256 x 1K = 256 K** |
| **p. ind. dobles** | **$256^2$ x 1K = 65538 K** |
| **p. ind. triples** | **$256^3$ x 1K ≈ 16G** |
| **total** | **16 G** |

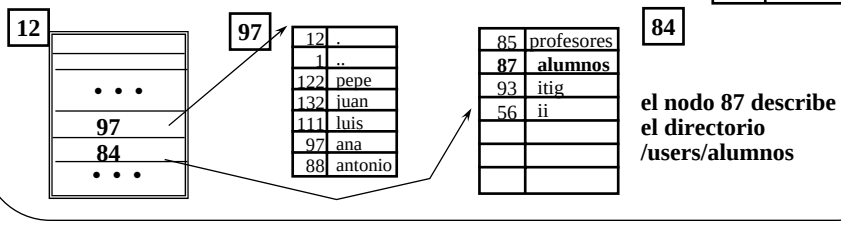
## Almacenamiento indexado

**Ejemplo:** **Obtener el fichero de nombre** **/users/alumnos/so/prueba.1**

**1. Obtener el nodo-i del raiz ("/") que se encuentra en una posición fija del disco.**




**2. Obtener los bloques de datos para averiguar el siguiente nivel.**

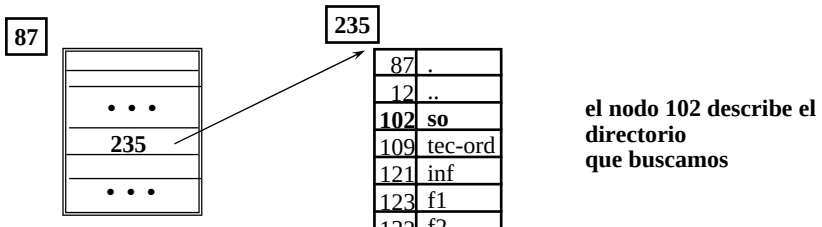




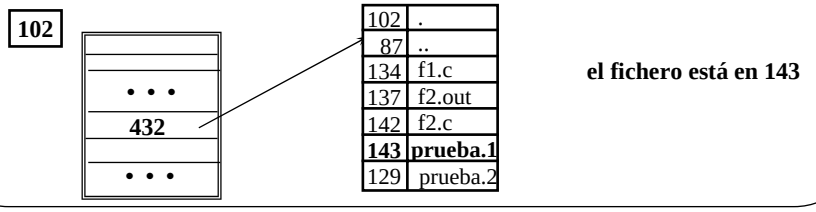

## Almacenamiento indexado

**Ejemplo (cont):**

**3. Se repite hasta encontrar el directorio último**



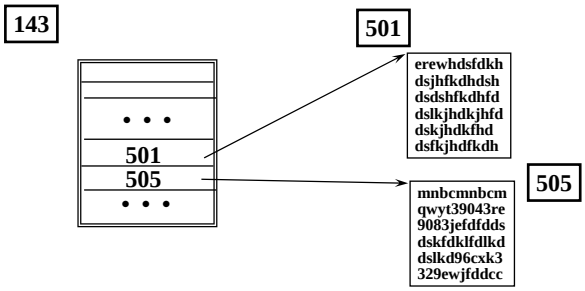

**4. Búsqueda del fichero**



## Almacenamiento indexado

# Ejemplo (cont.)

**5. Acceso al fichero de datos**






## Seguridad y Protección en los Sistemas de Ficheros

- La información del sistema debe ser protegida de daños físicos (fiabilidad) y acceso inadecuado (protección).
- El mecanismo de protección debe ofrecer un acceso controlado a los ficheros.
  - Controla operaciones sobre ficheros
    - **lectura/escritura de ficheros**
    - **ejecución: carga de un fichero en memoria para ejecutarlo**
    - **adición: añadir información al final de un fichero.**
    - **eliminación: borrar fichero y liberar su espacio en disco**
    - **otras: renombrar, copiar, editar… etc.**
  - Controla operaciones sobre directorios
    - **creación y eliminación de ficheros en el directorio**
    - **permitir listar el contenido de un directorio**
- Se puede proteger la ruta de acceso a un fichero o directorio.
  - Grafos acíclicos: distintos tipos de acceso en función de la ruta

## Métodos de protección

- **Protección asociada al nombre del fichero.**
  - Si no se conocen los nombres de los ficheros de un usuario (acceso a directorio denegado) no podré acceder a sus ficheros.
- **Protección por contraseña.**
  - Se asocia una contraseña a cada fichero.
    - Problema:
      - demasiadas contraseñas; puedo usar una para todos pero tengo menos protección.
      - es una protección muy fuerte TODO o NADA.





- **Protección por lista de acceso (NT).**
  - El acceso depende de la identidad del usuario.
  - Al fichero o directorio se le asocia una lista de acceso que especifique los permisos de cada usuario.
  - Cuando el usuario quiere acceder, el S.O. comprueba la lista y permite o deniega el acceso.

**PROBLEMA:**

- El tamaño de la lista puede variar y ser demasiado grande.
- El tamaño de la entrada en el directorio ya no es fijo.

## Seguridad y Protección en los Sistemas de Ficheros

- **Protección por grupos de acceso (UNIX).**
  - Es como las listas de acceso pero condensada.
  - Se clasifica a los usuarios en relación al fichero como:
    - Dueño: usuario que lo creo.
    - Grupo: usuario que necesitan un acceso similar al fichero.
    - Universo: el resto de usuarios.
  - Es necesario controlar que usuarios pertenecen a un grupo.
    - Por ejemplo en UNIX, los grupos son creados por el root. Hay otros sistemas en los que no hay control.
  - UNIX: Tres campos (cada uno 3 bits) que indican tipo de acceso: uno para dueño, otro grupo, otro resto de usuarios (9bits)

**_rwxrwxrwx indica todos los permisos**

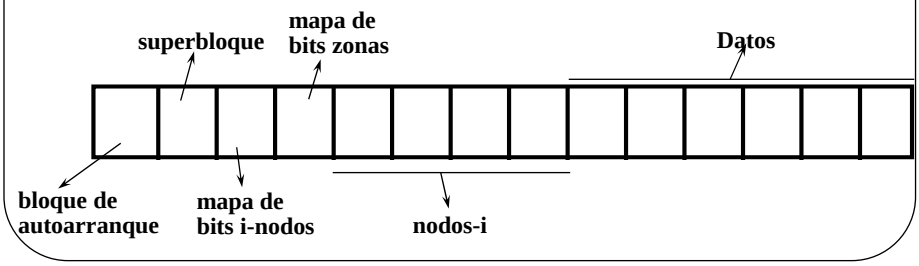


## Ejemplo: Organización de un dispositivo en MINIX

### Organización de un dispositivo

- **Tamaño del bloque: (Tiempo de acceso, aprovechamiento)**
  **MINIX -> 1K**
- **Hay que controlar los Bloques libres:**
  - lista enlazada y mapas de bits.

**Cualquier dispositivo orientado a bloques en MINIX se organiza como:**



## Ejemplo: Organización de un dispositivo en MINIX

### Descripción de los bloques

**Bloque de arranque: Al arrancar la máquina se ejecuta en ROM un programa de carga inicial. El programa accede a los dispositivos por defecto, carga el bloque de autoarranque en memoria y lo ejecuta. Este bloque permite la carga inicial del sistema operativo.**

**Superbloque: Descripción de la estructura del disco (tamaño de los elementos). Al arrancar el superbloque se carga en memoria.**

**Nodos-i: Reserva de varios bloques para almacenar los nodos-i. En discos de 360K se reservan 4 bloques (1K, 32 bytes/nodo-i) que dá lugar a 128 nodos-i (el primero no se utiliza).**

**Mapa de bits de nodos-i: estado de ocupación de los distintos nodos-i (8192 - 1) para representar los 127 nodos-i**

**Mapa de bits de zonas: (Zona = 2 bloques contiguos, n = 1). estado de las zonas (bloques del disco).**

**Bloque de datos: Bloques de datos (ficheros de datos) + bloques de punteros a otros bloques.**



## Ejemplo: Organización de un dispositivo en MINIX

### Superbloque

| | |
|---|---|
| disco + memoria | N. de nodos |
| | N. de zonas |
| | N.B. del M. bits de nodos-i |
| | N.B. del M. bits de zonas |
| | Primera zona de datos |
| | Log2 (zona/bloque) |
| | Max. tamaño de fichero |
| | Número mágico |
| sólo memoria | Punt. a M.bits nodos-i |
| | ••• |
| | Punt. a Mbits zonas |
| | ••• |
| | N. de disp. de superbloque |
| | Nodo-i del sist. de fich. montado |
| | Nodo-i en el que se monta |
| | Última actualización |
| | Indicador de sólo lectura/sucio |

```
EXTERN struct super_block {
  inode_nr s_ninodes;  /* # usable inodes on the minor device */
  zone_nr s_nzones;  /* total device size, including bit maps etc */
  unshort s_imap_blocks; /* # of blocks used by inode bit map */
  unshort s_zmap_blocks; /* # of blocks used by zone bit map */
  zone_nr s_firstdatazone; /* number of first data zone */
  short int s_log_zone_size; /* log2 of blocks/zone */
  file_pos s_max_size;  /* maximum file size on this device */
  short s_magic;  /* magic number to recognize super-blocks */

  /* These items are used when the super_block is in memory*/
  struct buf *s_imap[I_MAP_SLOTS]; /* p. to inode bit map */
  struct buf *s_zmap[ZMAP_SLOTS]; /* p. to zone bit map */
  dev_nr s_dev;   /* whose super block is this? */
  struct inode *s_isup;  /* inode for root dir of mounted file sys */
  struct inode *s_imount; /* inode mounted on */
  real_time s_time;  /* time of last update */
  char s_rd_only;  /* set to 1 iff file sys mounted read only */
  char s_dirt;   /* CLEAN or DIRTY */
} super_block[NR_SUPERS];
```