1. Sistema Operativo Unix

1.1 Introducción al S.O. Unix y su entorno

1.2 Subsistema de Archivos

1.3 Subsistema de Procesos

1.4 Políticas de Gestión de Memoria

1.2 Subsistema de archivos: Introducción

- **i-nodo**: representación interna de un archivo
- Un archivo tiene asociado un único i-nodo, aunque éste puede tener distintos nombres (**enlaces**)
- Si un proceso:
  - Crea un archivo ➔ se le asigna un i-nodo
  - Referencia a un archivo por su nombre ➔ se analizan permisos y se lleva el i-nodo a memoria principal

Introducción al subsistema de archivos (y II)

- Estructuras de datos en memoria principal relacionadas con los archivos:

  (1) **Tabla de i-nodos**: el núcleo lee del sistema de archivos el i-nodo cuando se opera con él

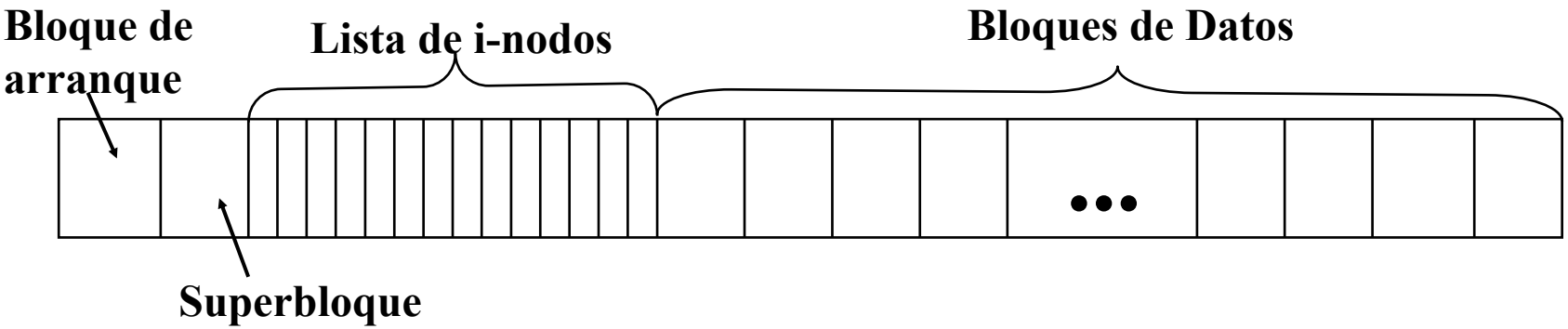
  (2) **Tabla de archivos**: mantiene información del puntero de lectura/escritura y los permisos de acceso del proceso al archivo

  (3) **Tabla de descriptores de archivos**: identifica los archivos abiertos por el proceso

- 1 y 2 son estructuras globales, 3 es una estructura local a cada proceso

Estructura en disco del sistema de archivos

- Un sistema de archivos es una secuencia de bloques lógicos con la siguiente estructura (***s5fs***):
  - **Bloque de arranque**: primer sector, puede contener código de arranque para inicializar el SO
  - **Superbloque**: estado del sistema de archivos
  - **Lista de i-nodos**: tamaño estático especificado en la configuración del sistema de archivos
  - **Bloques de datos**: para archivos y para administración



Contenido de un i-nodo

- **Identificador del propietario** del archivo: UID, GID
- **Tipo de archivo** (regular, directorio, dispositivo, cauce).

Si es 0 ➔ el i-nodo está libre

- **Permisos de acceso**
- **Tiempos de acceso**: última modificación, último

acceso y última vez que se modificó el i-nodo

- **Contador de enlaces**
- Tabla de contenidos para las **direcciones de los datos**

en disco del archivo

- **Tamaño**

Contenido de un i-nodo de la tabla de i-nodos

- El núcleo mantiene en la tabla de i-nodos, además del contenido del i-nodo, los siguientes campos:
  - El **estado** del i-nodo en memoria, indicando si:
    - » el i-nodo está bloqueado
    - » un proceso está esperando a que el i-nodo se desbloquee
    - » la representación en memoria del i-nodo difiere de la copia en disco por cambio en los datos del i-nodo
    - » la representación en memoria del archivo difiere de la copia en disco por cambio en los datos del archivo
    - » el archivo es un punto de montaje

Contenido de un i-nodo de la tabla de i-nodos (y II)

– El **número de dispositivo lógico** del sistema de archivos que contiene al archivo

– El **número de i-nodo**

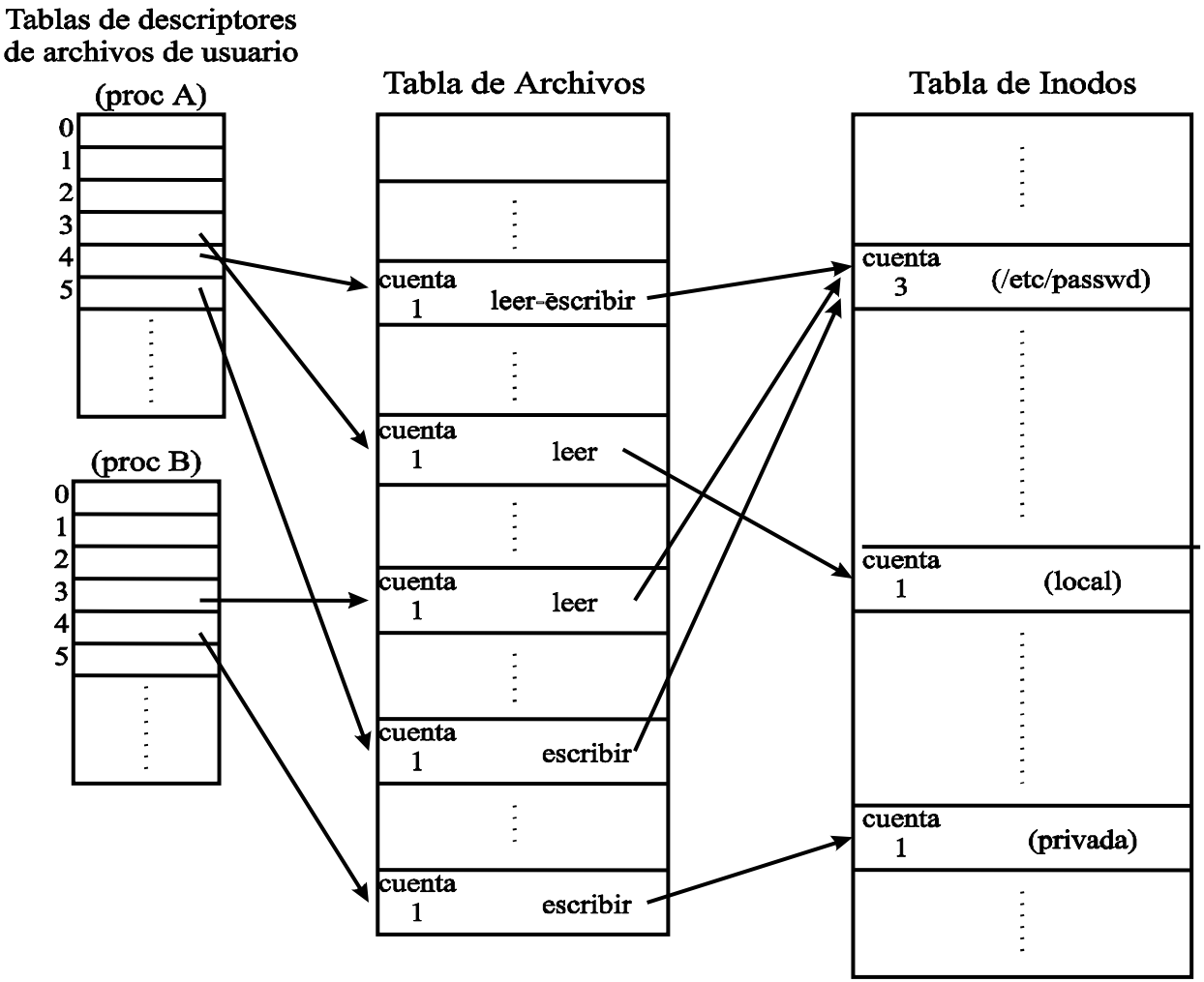
– **Punteros a otros i-nodos** en memoria: se usa una estructura hash para enlazar los i-nodos cuya clave de acceso se basa en el número de dispositivo lógico y el número de i-nodo. Existe una lista de i-nodos libres.

– **Contador de referencias**: número de referencias actuales al i-nodo (p.e. cuando varios procesos abren el mismo archivo). Un i-nodo sólo estará en la lista de i-nodos libres cuando su contador de referencias sea 0.

Tabla de archivos y Tabla de descriptores de archivos

- **Tabla de archivos**, contenido de cada entrada:
  - puntero al i-nodo correspondiente de la tabla de i-nodos
  - puntero de lectura/escritura
  - permisos de acceso
  - modo de apertura del archivo
  - contador de entradas de las tablas de descriptores de archivos asociados con esta entrada
- **Tabla de descriptores de archivos**, contenido de cada entrada:
  - puntero a la entrada de la tabla de archivos correspondiente

Estructuras de datos después de que dos procesos abran archivos



El sistema abre para cada proceso tres archivos por defecto (primeras entradas de la tabla de descriptores de archivos):

- **stdin** (teclado)
- **stdout** y **stderr** (pantalla)

Algoritmos a bajo nivel del S.A.

– **iget** = devuelve un i-nodo de la tabla de i-nodos identificado previamente, si no está, lo carga en la tabla
– **iput** = libera un i-nodo de la tabla de i-nodos
– **namei** = convierte un *pathname* a un número de i-nodo
– **alloc** y **free** = asignan y liberan bloques de disco
– **ialloc** e **ifree** = asignan y liberan i-nodos de disco
– **bmap** = traducción de direcciones lógicas a físicas

<table>
<tr><td>namei</td><td rowspan="2">alloc free</td><td rowspan="2">ialloc ifree</td></tr>
<tr><td>iget iput bmap</td></tr>
<tr><td colspan="3">algoritmos de asignación de la buffer caché</td></tr>
</table>

Algoritmo iget: acceso a i-nodo en memoria

Entrada: número de i-nodo de un SA
Salida: i-nodo bloqueado

```
while (no hecho) {
        if (i-nodo en tabla de i-nodos) {
                if (i-nodo bloqueado) {
                        bloquear (hasta i-nodo desbloqueado);
                        continue;  /* vuelve al while */
                }
                bloquear (i-nodo);
                if (i-nodo en la lista i-nodos libres)
                        eliminarlo de la lista de libres;
                contador_referencias ++;
                return (i-nodo);
        }
```

Algoritmo iget: acceso a i-nodos en memoria (y II)

```
        else /*i-nodo no está en la tabla de i-nodos*/ {
                if (no hay i-nodos en lista i-nodos libres)
                        return (error);
                liberar i-nodo de la lista i-nodos libres;
                actualizar nº i-nodo y Sistema de Archivos;
                colocar el i-nodo en la entrada correcta de
                        la tabla hash;
                leer i-nodo de disco; /* bread */
                inicializar i-nodo; /*contador_referencias=1*/
                return (i-nodo);
        }
} /* fin de iget */
```

Algoritmo iput: liberación de i-nodos en memoria

Entrada: puntero al i-nodo en memoria (tabla de i-nodos)

Salida: nada

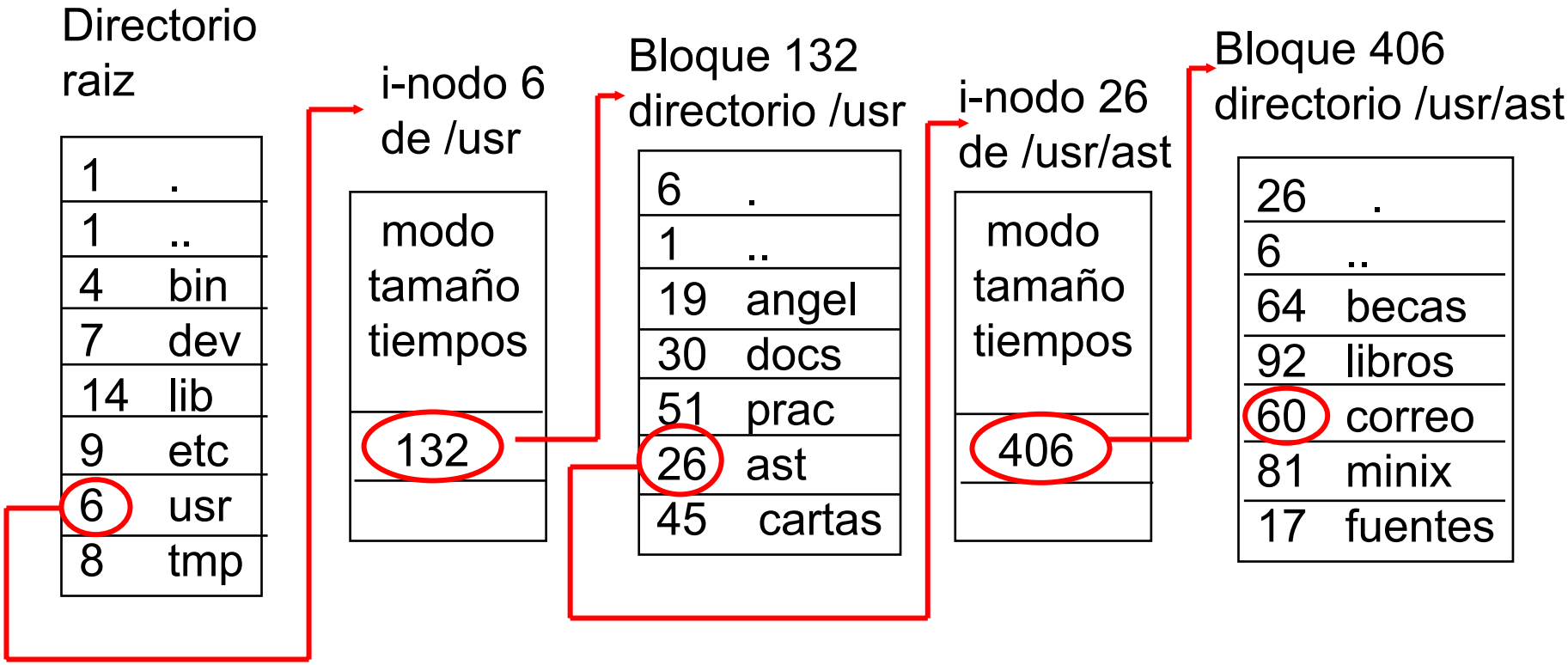
```
{
bloquear (i-nodo);    /*si no lo está*/
  contador_referencias --;
  if (contador_referencias == 0)  {
      if (contador_enlaces == 0) {
            liberar bloques de disco del archivo;  /* free */
            poner tipo de archivo a 0;   /* i-nodo libre */
            liberar (i-nodo);  /* ifree */   }
      else {
            if (accedido(archivo) || cambiado(i-nodo) ||
                    cambiado(archivo) )
                actualizar i-nodo de disco;
      poner i-nodo en la lista de i-nodos libres;
      }
  }
desbloquear ( i-nodo);  } /* fin de iput */
```

Archivos regulares y directorios

- **Archivo regular**
  - secuencia de bytes, comenzando en el byte 0
  - los procesos acceden a los datos mediante un desplazamiento (**offset**)
  - los archivos pueden contener **agujeros** (lseek, write)
- **Directorios**
  - diferencias entre BSD y SVR4 solventadas utilizando una biblioteca estándar de manejo de directorios
  - un directorio es un archivo cuyos datos son secuencias de entradas. Dos entradas especiales son **.** y **..**
  - entradas de directorios vacías ➔ número i-nodo = 0
  - cada proceso tiene asociado un directorio actual
  - el i-nodo del directorio raíz se almacena en una variable global

Ejemplo de namei

- Supongamos el nombre de ruta */usr/ast/correo,* ¿cómo se convierte en un nº de i-nodo?



Contenido del superbloque

- Tamaño del sistema de archivos
- Número de bloques libres en el sistema de archivos
- Una lista de bloques libres disponibles y un índice al siguiente bloque libre de la lista de bloques libres
- Tamaño de la lista de i-nodos
- Número de i-nodos libres en el sistema
- Una lista de i-nodos libres y un índice al siguiente i-nodo libre de la lista de i-nodos libres
- Además, cuando está en memoria tiene:
  - Campos de bloqueos para las listas de bloques e i-nodos libres
  - Un campo indicando que el superbloque ha sido modificado

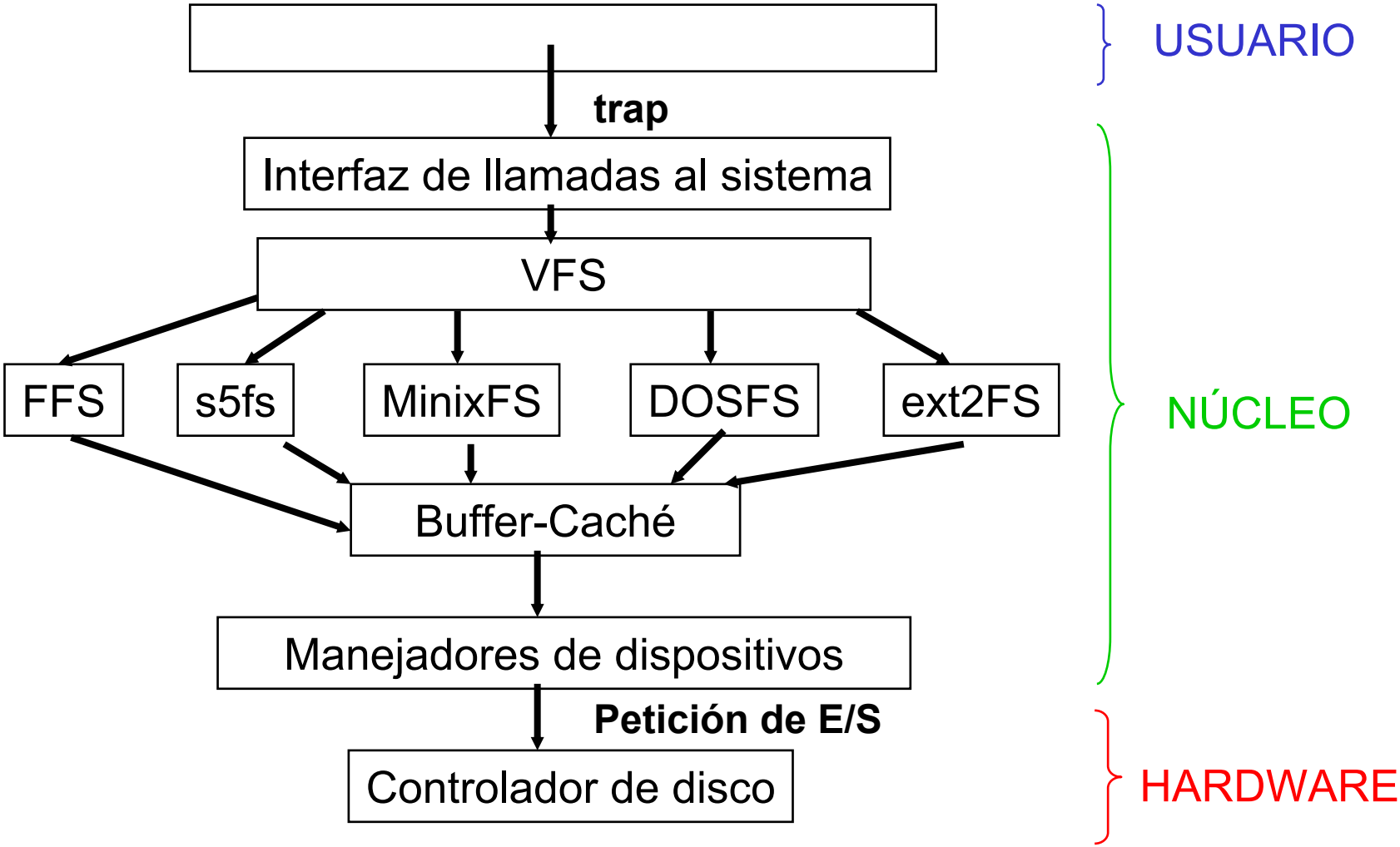
Problemas de s5fs

- Sólo una copia del superbloque ➔ **menor fiabilidad**
- Los i-nodos se asignan aleatoriamente ➔ **varios accesos** para archivos en un mismo directorio
- El i-nodo no se asocia de ninguna forma a sus bloques de datos
- La asignación de bloques de datos no es óptima, salvo inicialmente (después de crear el S.A.)
- Tamaño de bloque (de 1 a 4KB)
- Límite en número de i-nodos (2 bytes) y en nombre de archivos (14 caracteres)

ufs - unix file system

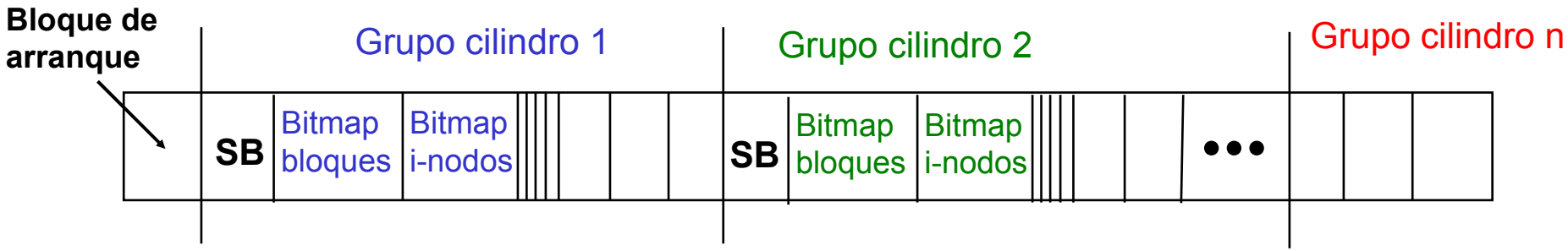
- **ufs** = interfaz vnode/vfs (virtual node/ virtual file system) + implementación FFS (Fast File System)
- **Objetivos**:
  - Soporte para distintos tipos de sistemas de archivos simultáneamente
  - Diferentes particiones con diferentes sistemas de archivos
  - Transparencia en el acceso a un sistema de archivos remoto
  - Añadir nuevos tipos de sistemas de archivos de forma modular

vnode/vfs


USUARIO
trap
Interfaz de llamadas al sistema
VFS
FFS
s5fs
MinixFS
DOSFS
ext2FS
NÚCLEO
Buffer-Caché
Manejadores de dispositivos
Petición de E/S
Controlador de disco
HARDWARE

FFS - Fast File System

- Partición dividida en uno o más grupos consecutivos de cilindros
- La información del superbloque tradicional se divide en:
  - Superbloque FFS: número, tamaño y localización de grupos de cilindros, tamaño bloque, nº total de i-nodos y bloques
  - Cada grupo de cilindros tiene su propia lista de i-nodos libres y de bloques libres



FFS (y II)

- Cada grupo de cilindros contiene una copia duplicada del superbloque
- El tamaño mínimo del bloque es **4KB** ➔ no se necesita tercer nivel de indexación en el i-nodo
- El bloque se puede dividir en fragmentos direccionables y asignables ➔ lista de bloques libres mediante mapa de bits
- Sólo los bloques directos del i-nodo pueden contener fragmentos
- Entradas variables en directorios y enlaces simbólicos

FFS (y III)

- **Políticas de asignación**:
  - Intenta situar i-nodos de archivos de un único directorio en el mismo grupo de cilindros
  - Crea un directorio en un grupo diferente al de su padre ➔ distribuye datos uniformemente
  - Intenta asignar bloques de datos en el mismo grupo que su i-nodo correspondiente
  - Cambia de grupo cuando se alcancen ocupaciones preestablecidas

Llamadas al Sistema

Devuelven un descriptor	Usan namei	Asignan y liberan i-nodos	Atributos de archivo	E/S archivos	Estructura S.Archivos	Manipulan árbol S.A.
open creat dup pipe close	open creat chdir chroot chown chmod stat link unlink mknod mount umount	creat mknod link unlink symlink	chown chmod stat fstat	read write lseek	mount umount	chdir chown

Llamada al sistema **open**

- Sintaxis:

```
da = open (<camino>, <opciones>, [<permisos>])
```

opciones: modo de apertura (O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT, ...)

- Pasos:
  - Convierte el camino en número de i-nodo (**namei**)
  - Si (archivo no existe || no permisos) return (error)
  - Asigna una entrada en la tabla de archivos para este i-nodo e inicializa el contador y offset (por defecto, 0)
  - Asigna una entrada en la tabla de descriptores de archivos y la enlaza con la entrada de la tabla de archivos anterior
  - Desbloquear i-nodo
  - Devolver ***da*** (descriptor de archivo)

Cauces (*pipes*)

- Permiten la transferencia de datos y la sincronización entre procesos. Capacidad limitada y gestión FIFO
- En la última implementación son bidireccionales y soportan comunicación *full-duplex*

sin nombre

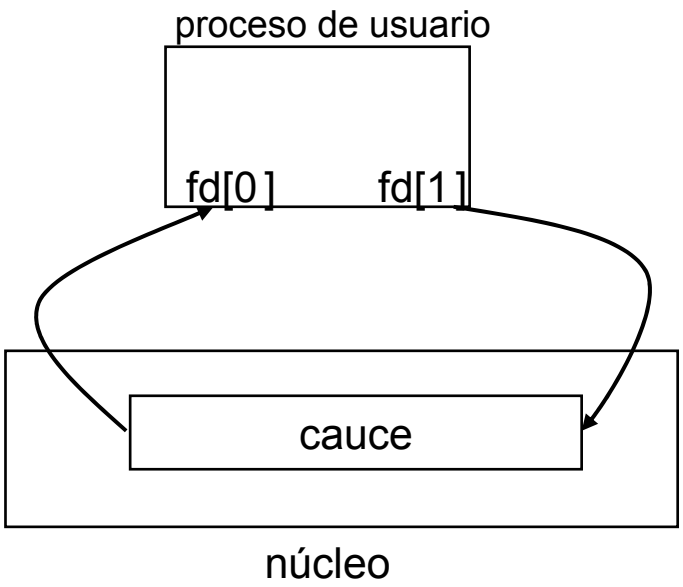
- se crean con **pipe**
- sólo el proceso que lo crea y sus descendientes pueden compartirlo
- son transitorios
- no tienen asociado un nombre de archivo
- tienen asociado un i-nodo en la Tabla de i-nodos

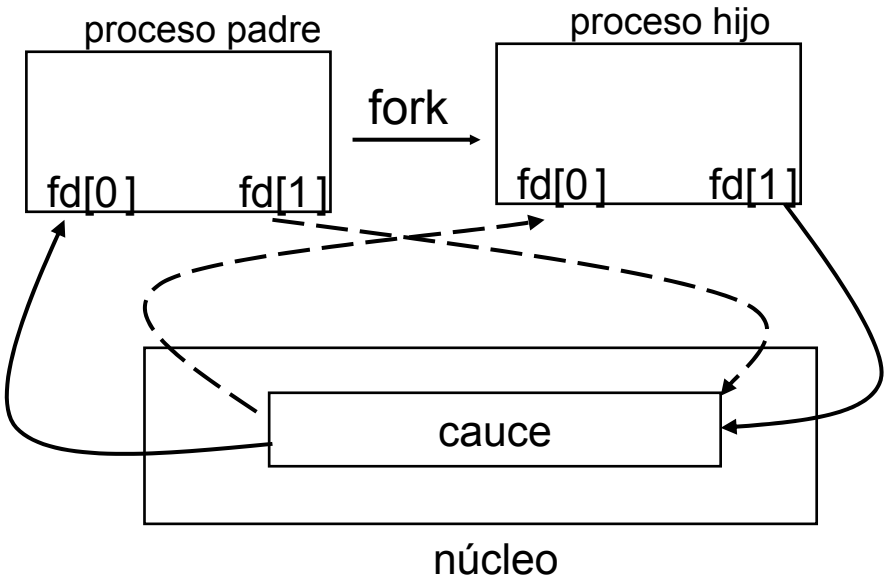
con nombre

- se crean con **mknod**
- todos los procesos con privilegios tienen acceso a él
- no son transitorios, ocupan permanentemente una entrada en un directorio
- tienen asociado un nombre de archivo
- tienen asociado un i-nodo en la Tabla de i-nodos y en disco

Cauces (y II)

int pipe (int fildes[2]) ➔ devuelve 0, si éxito, -1 si falla





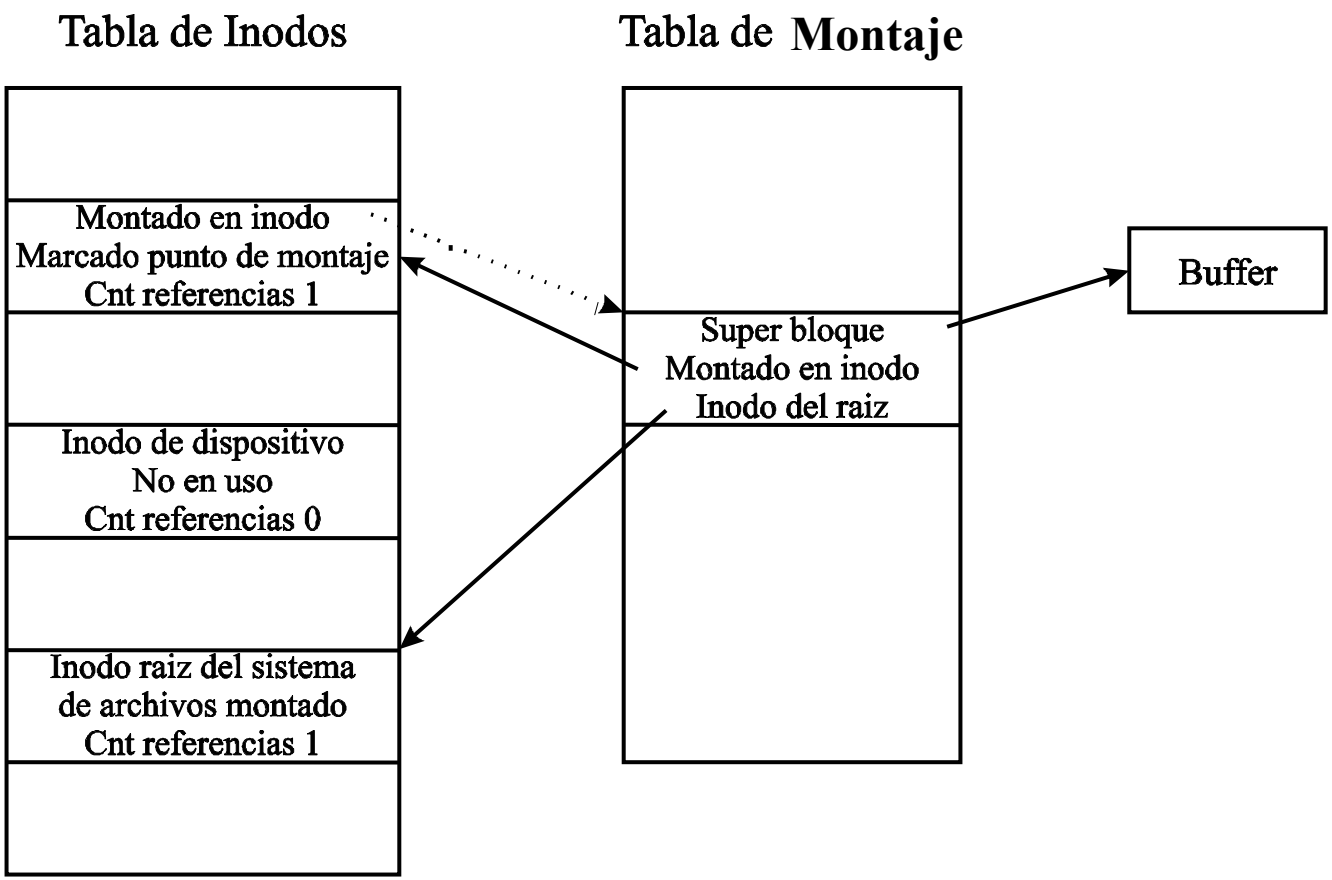
Montaje y desmontaje de un sistema de archivos

- La llamada al sistema **mount** conecta un sistema de archivos al sistema de archivos existente y la llamada **umount** lo desconecta

```
mount (<camino_especial>, <camino_directorio>, <opciones>)
```

- El núcleo tiene una tabla de montaje con una entrada por cada sistema de archivos montado:
  - número de dispositivo que identifica el SA montado
  - puntero a un buffer que contiene una copia del superbloque
  - puntero al i-nodo raíz del SA montado
  - puntero al i-nodo del directorio punto de montaje

Montaje y desmontaje de un sistema de archivos (y II)



Estructuras de Datos después de Montar

Buffer Caché

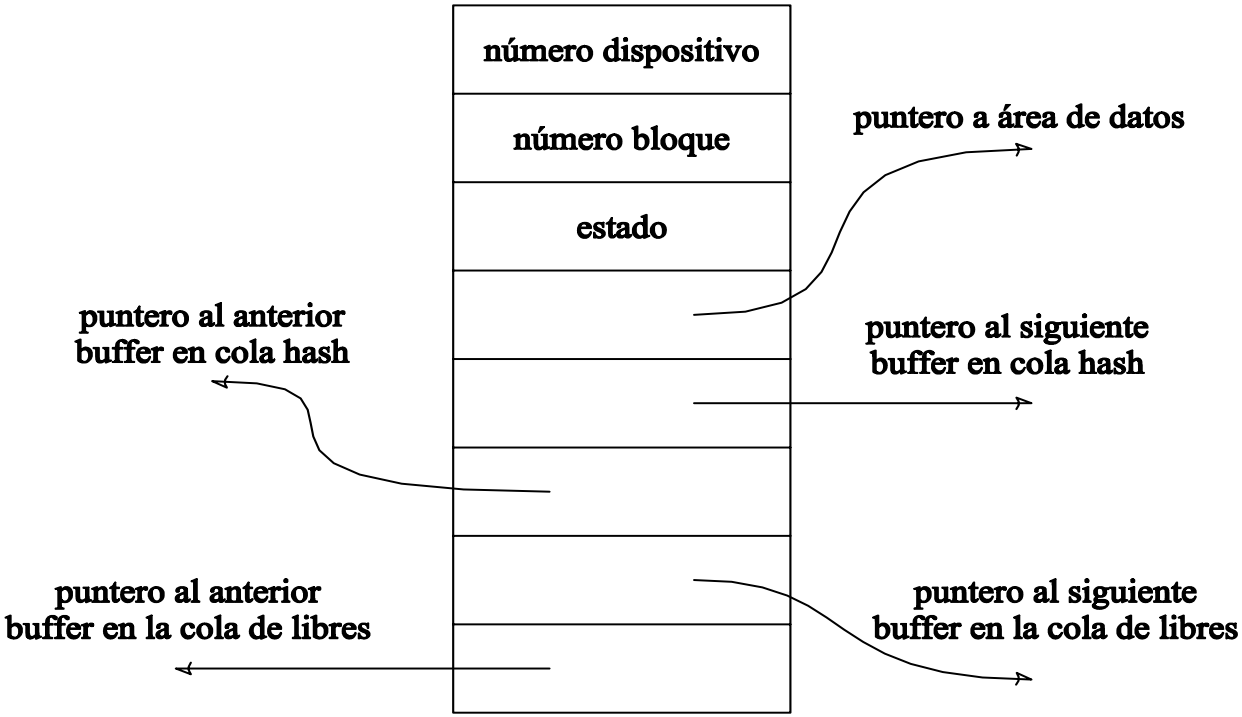
- Limita los accesos a disco aumentando la eficiencia
- Mantiene en memoria principal un depósito de datos interno que contiene los datos de los bloques de disco recientemente usados
- Cuando se leen datos del disco, el núcleo tiende a leerlos de la buffer caché (igual para escritura):
  - Si los datos están, no se tiene que acceder a disco
  - Si los datos no están se accede a disco y se introducen en la caché
- El tamaño de la buffer caché se puede determinar durante la inicialización del sistema
- En las implementaciones actuales la buffer caché sólo se utiliza para los metadatos

Partes de un buffer

1. **Copia de un bloque lógico de disco**. Un bloque de disco no se puede encontrar en mas de un buffer.
2. **Cabecera del buffer** que lo identifica. Contenido:
   - Número de dispositivo
   - Número de bloque
   - Puntero a la copia del bloque de disco
   - Estado, combinación de las siguientes condiciones:
     - » bloqueado (ocupado)
     - » contiene datos válidos
     - » **escritura retardada** (el núcleo debe escribir el contenido del buffer antes de reasignarlo)
     - » se está leyendo/escribiendo el contenido del buffer
     - » un proceso está esperando a que el buffer se libere

Partes de un buffer (y II)

- Punteros utilizados por los algoritmos de asignación para mantener la estructura del depósito de bufferes.



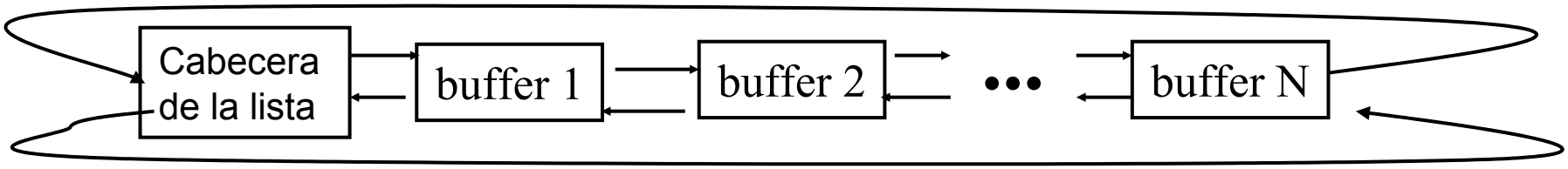
Cabecera del *Buffer*

Estructura de la buffer caché

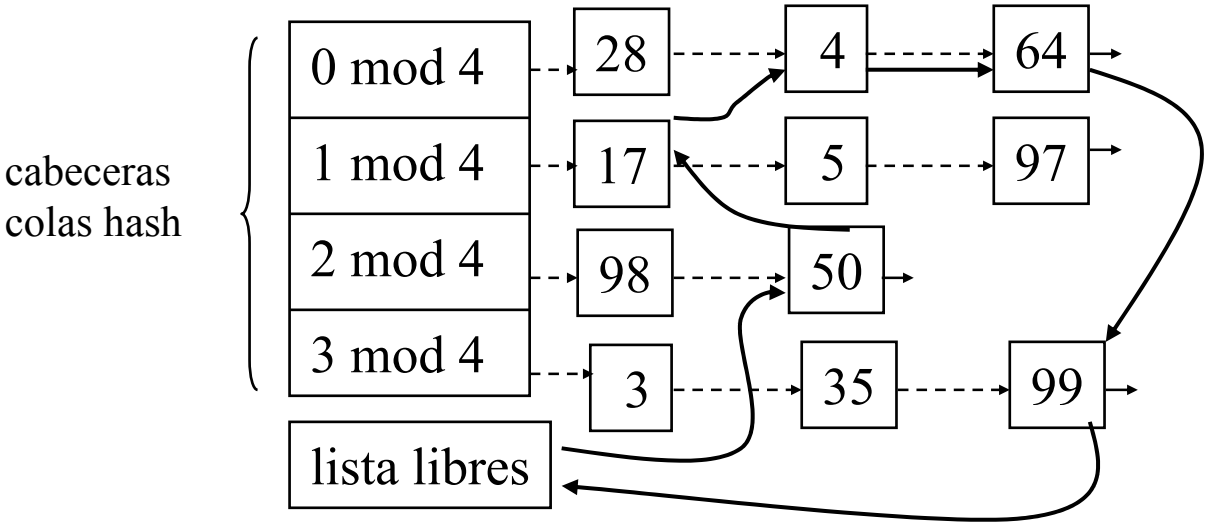
- Se gestiona mediante el algoritmo **LRU**
- **Organización**:
  - lista circular doblemente enlazada de bufferes libres
  - varias colas separadas (listas doblemente enlazadas). Utiliza una función *hash* basada en **nºdispositivo + nºbloque**
- El nº de elementos en una cola hash varía durante la vida del sistema
- Cada buffer está en una única cola hash aunque también puede estar en la lista de bufferes libres
- La posición del buffer dentro de la cola hash no es relevante

Estructura de la buffer caché (y II)

Inicialmente, todos los bufferes están libres



Después de un tiempo ...



- son **listas doblemente enlazadas**
- en el ejemplo sólo usamos el nº de bloque por simplicidad

Algoritmos a bajo nivel para la buffer caché

- **getblk** ➔ lo usan los algoritmos de lectura/escritura para asignar bufferes del depósito
- **brelse** ➔ libera un buffer
- **bread** ➔ lee un bloque de disco (usa getblk)
- **breada** ➔ lecturas asíncronas adelantadas de bloques de disco (usa getblk, brelse)
- **bwrite** ➔ escribe los contenidos de un buffer a un bloque de disco (usa brelse)

Escenarios para recuperar un buffer

1. El núcleo encuentra el bloque en su cola hash y su buffer está en la lista de libres

- marca el buffer como ocupado
- saca el buffer de la lista de libres

2. El núcleo no encuentra el bloque en su cola hash y hay bufferes en la lista de libres

- ¿primer buffer de la lista de libres marcado como escritura retardada?

  no ➔ lo marca como ocupado, lo saca de la cola hash actual y lo inserta en la cola hash apropiada

  si ➔ inicia su escritura asíncrona (cuando termine volverá a la cabeza de libres) y busca otro en la lista libres

Escenarios para recuperar un buffer (y II)

3. El núcleo no encuentra el bloque en la cola hash y la lista de libres está vacía

- bloquear al proceso en espera de un buffer libre

4. El núcleo encuentra el bloque en la cola hash pero está ocupado

- bloquear al proceso hasta que se libere
- **!cuidado!** cuando el proceso se desbloquee debe comenzar la comprobación desde el principio

Ventajas y desventajas

- **Ventajas**
  - acceso a disco uniforme, código más modular
  - reducción del trafico de disco, incrementando el rendimiento y decrementando el tº de respuesta
  - los algoritmos ayudan a asegurar la **integridad del sistema** ➔ secuencializan el acceso a los datos
- **Desventajas**
  - vulnerabilidad ante caídas del sistema
  - copia adicional de los datos en el proceso de usuario y en la caché. Solución en sistemas actuales ➔ **archivos mapeados**