



Caches de disco

Page cache & buffer cache

Diseño de Sistemas Operativos

Grado en Ing. de Computadores, 3er Curso

Depto. de Arquitectura de Computadores
Universidad de Málaga

● ● ● | Contenidos

- Discos: dispositivos de bloques
- Caches de disco
- Page Cache (ficheros)
- Buffer Cache (bloques)
- Lectura/escritura buffers (bloques)
- Escritura de páginas a disco
- Planificación de E/S
 - Peticiones de E/S
 - “Ascensor” de linux

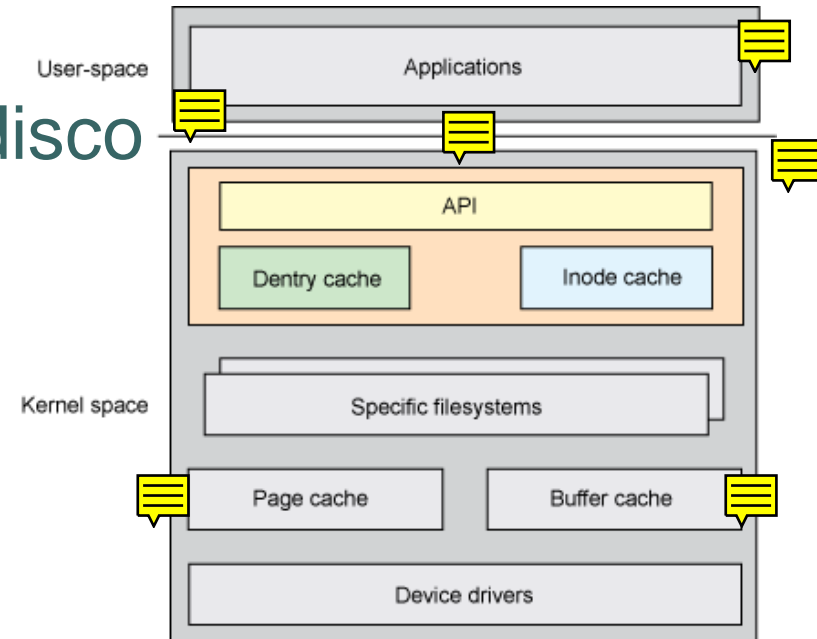
● ● ● | Dispositivos de bloques

- Discos duros, Blue-ray, memoria flash, etc.
 - Se accede a bloques de datos de tamaño fijo
 - Acceso aleatorio (random access)
 - Se accede directamente a los bloques de disco (raw) o mediante un sistema de ficheros
 - Operaciones: open, read, write, seek, sync,....
 - Es posible mapear en memoria un fichero y acceder como a un array de memoria
- Dispositivos de caracteres son más sencillos de gestionar (teclado, puertos serie, ...)
 - se accede a un stream linealmente

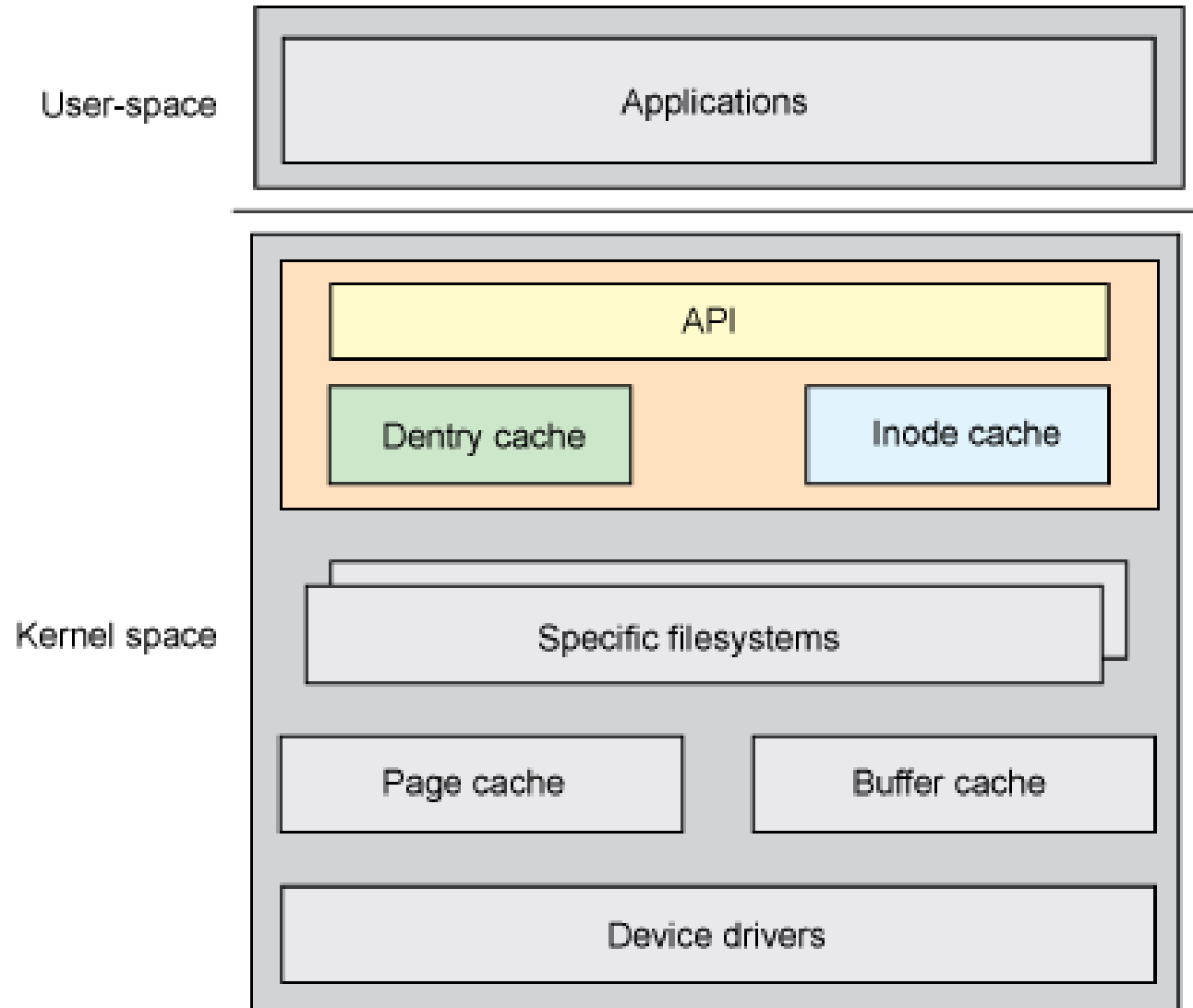
● ● ● | Caches para acceso a disco

● Cache de disco

- El sistema mantiene en memoria datos del disco
- Los datos son accedidos rápidamente sin tener que ir al disco
- Localidad en accesos a disco
 - Del mismo proceso
 - Procesos diferentes
- Crucial para rendimiento



● ● ● | Caches para acceso a disco



● ● ● | Page cache (ficheros)

- Es la cache de disco principal en linux
- Se usa para **leer y escribir** en disco
 - Páginas completas de memoria
 - Corresponden a trozos de ficheros
- Las páginas se mantienen en la cache mientras se pueda
- Se escriben a disco lo más tarde posible
 - Escritura diferida
- Es posible lectura adelantada en ficheros
 - Precarga

● ● ● | Page cache

- Contenido de la page cache
 - Páginas de ficheros
 - Páginas de directorios (son ficheros) 🗨
 - Páginas con datos de bloques de disco (no forman parte de un fichero)
 - Páginas intercambiadas a disco 🗨
- Cada página pertenece a un fichero (o dispositivo)
 - Propietario de la página
- Si un fichero se abre con `O_DIRECT` 🗨
 - No se usa el page cache
 - El kernel no controla el buffer (se hace en espacio de usuario)

● ● ● | Organización del page cache

- Estructura address_space
 - Cada propietario (fichero/dispositivo) mantiene un address_space con todas sus páginas del page cache
- Las páginas están enlazadas en un radix-tree
 - Almacenamiento compacto
 - Búsquedas rápidas

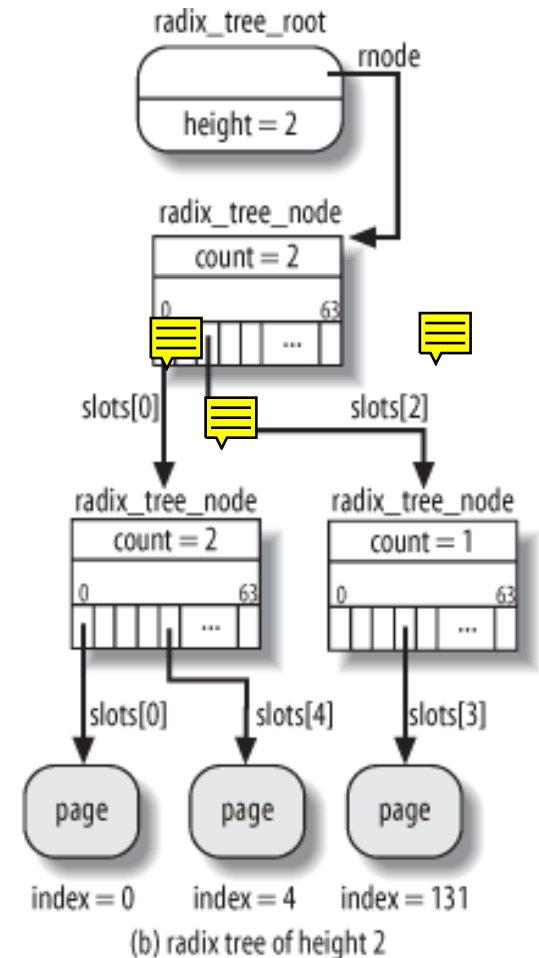
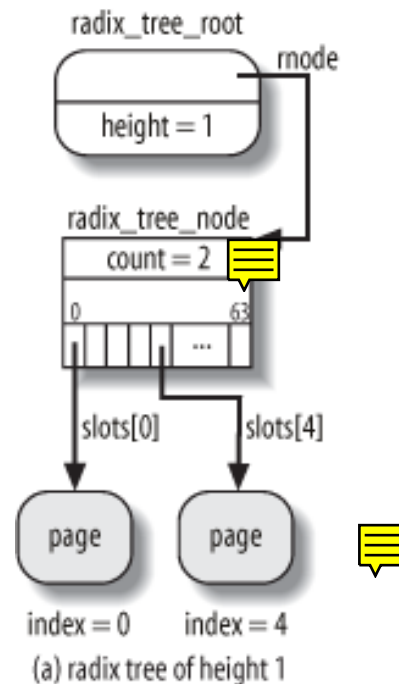
Radix tree

- Se indexa cada página en el radix tree usando su posición en disco dentro del fichero

- `add_to_page_cache()`
- `find_get_page()`

- Se usan etiquetas en los nodos para comprobaciones rápidas

- `PG_dirty`
- `PG_writeback`



● ● ● | Acceso a los bloques en el page cache

- Las páginas que contienen bloques se integran en el `address_space` de su propietario (dispositivo)
- Se indexan en el radix tree por página así que hay que traducir la dirección en bloques a su dirección de página
- La página (si existe) mantiene un puntero a la lista de `buffer_head` donde puede estar el bloque que buscamos

● ● ● | Lectura de bloques


- Cada página mantendrá buffers para bloques de igual tamaño
 - x86: página 4KB contiene de 1 a 8 bloques
- Leer el bloque que contiene el inodo de un fichero (1KB) en disco
 - El kernel reserva una página (4KB) para cuatro bloques
 - Lee cuatro bloques adyacentes en disco
 - Uno de ellos es el inodo que necesitamos

● ● ● | Escritura de páginas a disco

- La page cache se llena sin cesar
- Algunas páginas son modificadas PG_dirty
- El kernel retrasa la escritura lo máximo posible, hasta:
 - La page cache está muy llena
 - Hay demasiadas PG_dirty
 - Demasiado tiempo en PG_dirty
 - vamos a perder datos si no grabamos!
 - Un proceso pide el flush
 - sync() fsync() fdatasync()
- Para los buffers de bloques la página esta dirty si alguno de sus buffers lo está

● ● ● | Escritura de páginas a disco

- Kernel threads **pdflush**

- Escanean la page cache en busca de páginas dirty que volcar a disco 
- Entre 2 y 8 threads tabajando según la carga
- Los threads flusher se activan cada cierto tiempo para comprobar. También se lanza cuando la memoria está baja o cuando lo pide el usuario
 wakeup_flusher_threads()
- Modo laptop

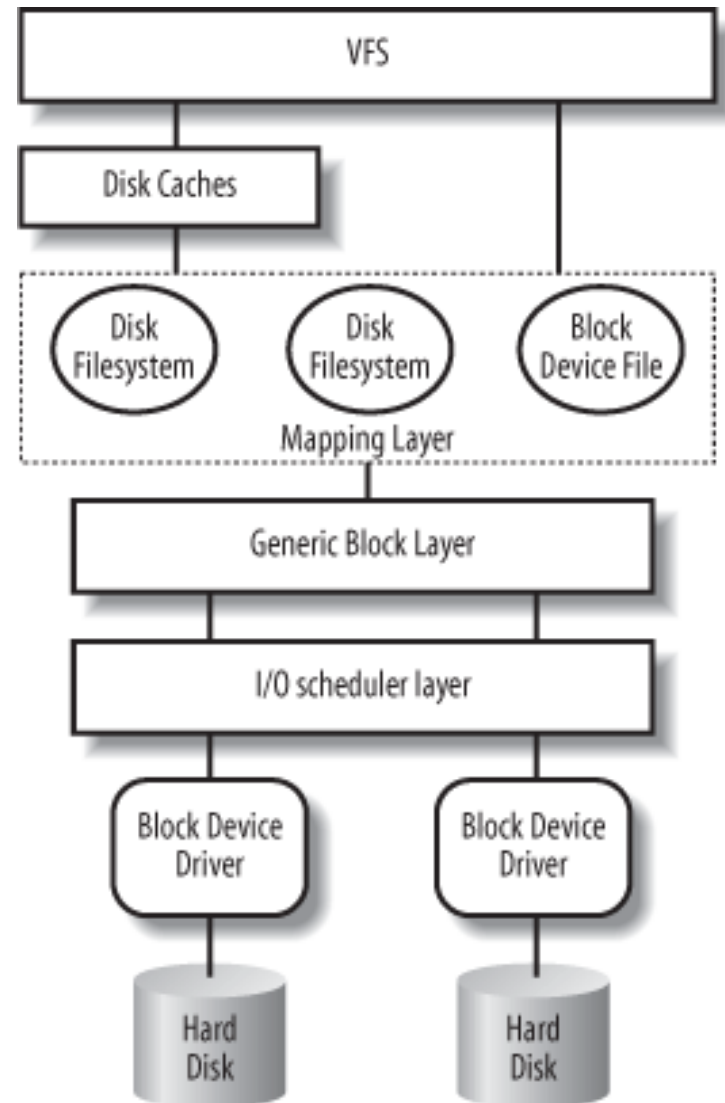
● ● ● | Configuración escritura págs.

| Variable | Descripción |
|---------------------------|--|
| dirty_background_ratio | Porcentaje de memoria total a partir del cual los threads flushers empiezan a escribir páginas dirty a disco |
| dirty_expire_centisecs | Antigüedad para que una página sea escrita a disco la próxima vez que se despierte el thread para comprobar (periódicamente) |
| dirty_ratio | Igual que el primero pero para un solo proceso |
| dirty_writeback_centisecs | Intervalo de tiempo que pasa entre cada activación (wake up) de los thread flushers |
| laptop_mode | 1 para activar el modo laptop 0 para desactivarlo |

Configurable en `/proc/sys/vm` o con `sysctl()`

● ● ● | Accediendo a un dispositivo por bloques

- De ficheros hay que pasar a bloques
- Para leer y escribir bloques se usan buffers
- También usamos una cache de buffers



● ● ● | Buffer cache (bloques E/S)

- Pool de buffers para transferencias de E/S en marcha y recientes (cacheadas)
- Cada buffer se compone de:
 - datos (el propio buffer) en una página del page cache
 - `buffer_head` (descriptor del buffer)
- Integrada en la page cache de linux desde kernel 2.4.10
- Buffers en páginas de memoria del *page cache*

● ● ● | Buffers en una página

- Sector

- Unidad de transferencia del disco

- Bloque

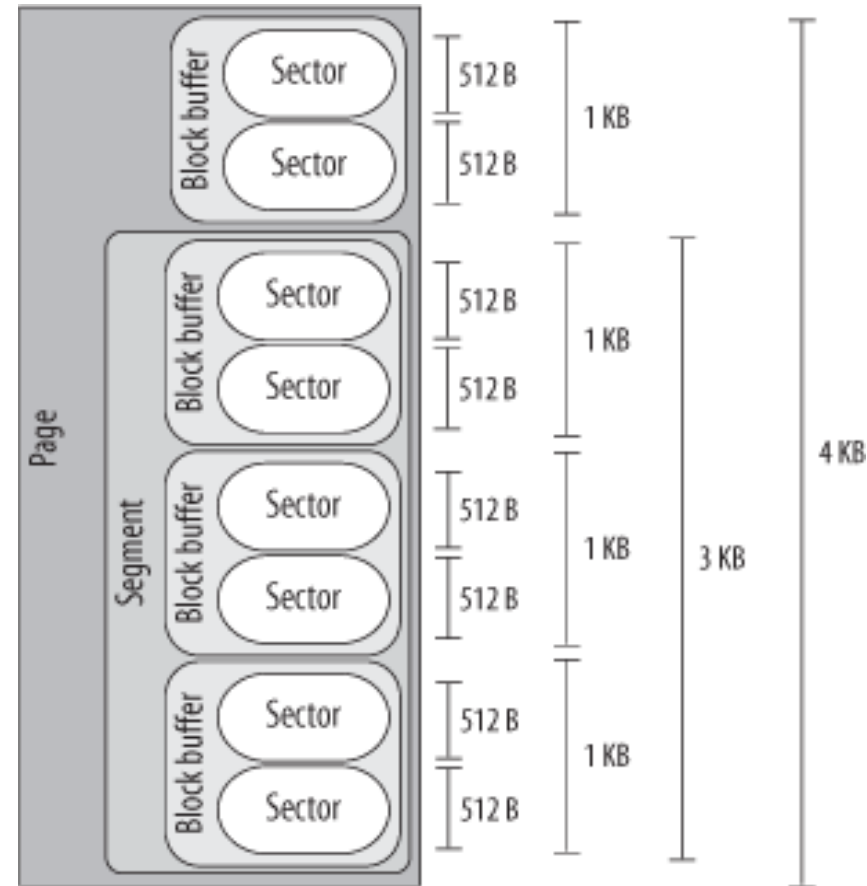
- Unidad de transferencia del sistema de ficheros

- Segmento

- Región contigua de memoria en una operación de DMA

- Página

- Unidad de asignación de memoria virtual

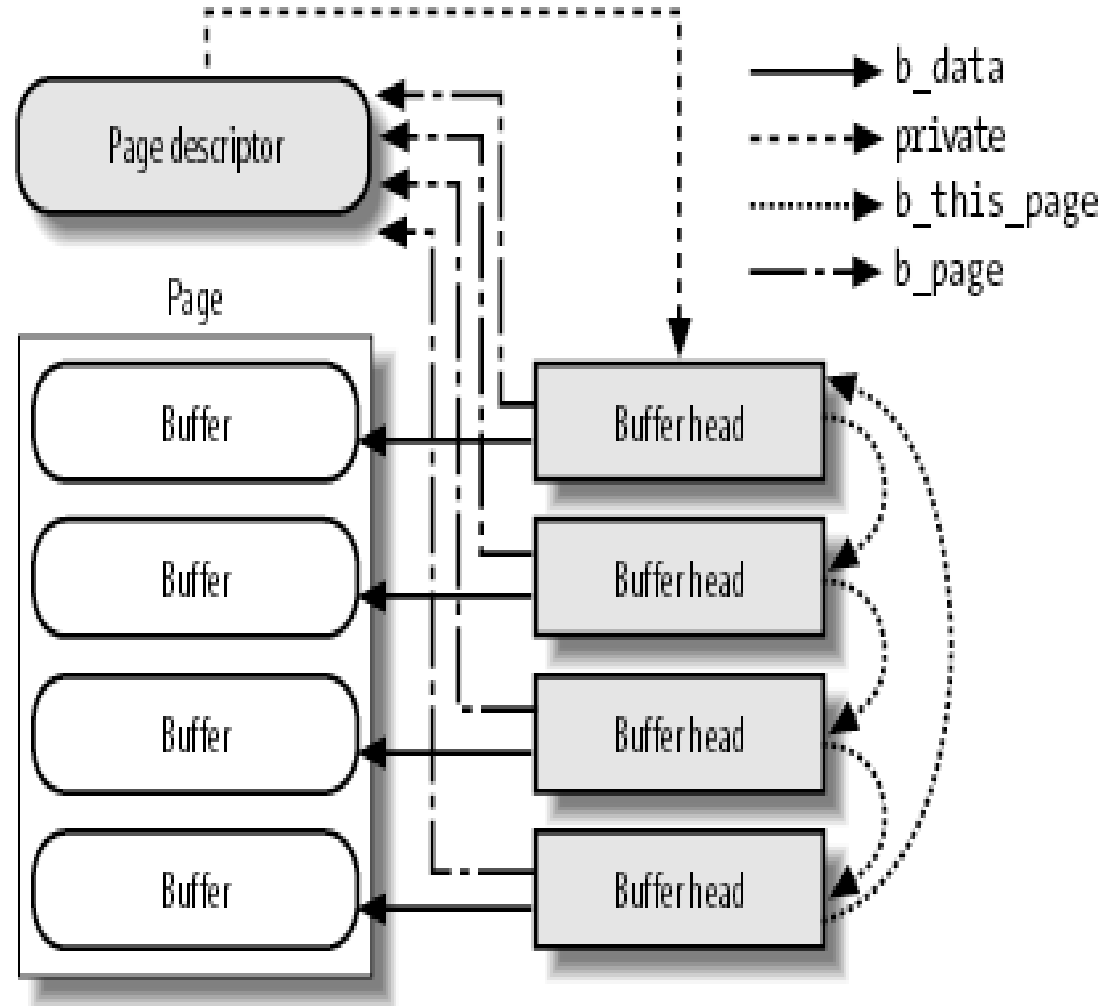


● ● ● | Buffer head

| Type | Field | Description |
|-----------------------|-----------------|---|
| unsigned long | b_state | Buffer status flags |
| struct buffer_head * | b_this_page | Pointer to the next element in the buffer page's list |
| struct page * | b_page | Pointer to the descriptor of the buffer page holding this block |
| atomic_t | b_count | Block usage counter |
| u32 | b_size | Block size |
| sector_t | b_blocknr | Block number relative to the block device (logical block number) |
| char * | b_data | Position of the block inside the buffer page |
| struct block_device * | b_bdev | Pointer to block device descriptor |
| bh_end_io_t * | b_end_io | I/O completion method |
| void * | b_private | Pointer to data for the I/O completion method |
| struct list_head | b_assoc_buffers | Pointers for the list of indirect blocks associated with an inode |

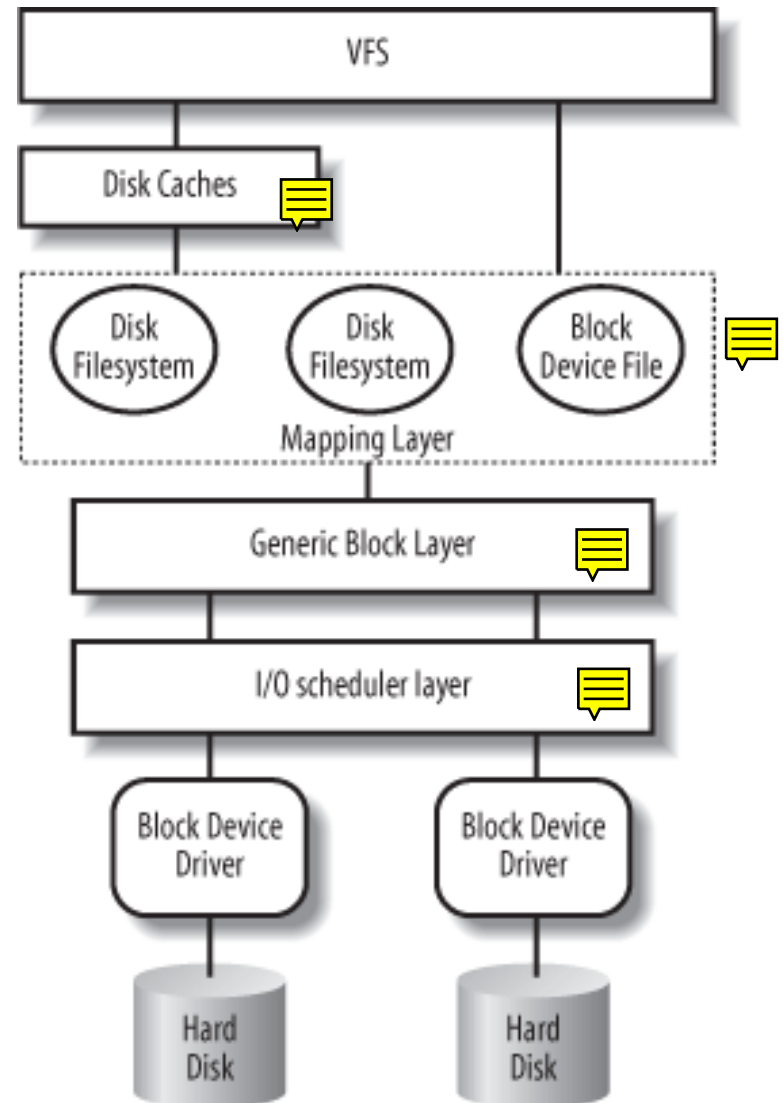
• • • | Páginas/buffers

- El kernel mantiene el conjunto de BHs de los buffers presentes en la cache



● ● ● | Accediendo a un dispositivo por bloques

- De ficheros hay que pasar a bloques
- Para leer y escribir bloques se usan buffers
- También usamos una cache de buffers



● ● ● | Peticiones de E/S

- La capa de acceso a bloques genérica (Generic Block Layer) resuelve las peticiones de lectura/escritura a disco
- Se basa en la estructura **bio** que describe la operación de E/S
 - Bloques de disco (dispositivo, bloques)
 - Segmentos de memoria (buffer_heads)
- Se realiza la petición de entrada/salida para la estructura **bio**
 - `generic_make_request()`

● ● ● | Planificador de E/S

- Cada petición de E/S contiene una estructura **bio** inicialmente
- No se ejecutan inmediatamente
- El planificador de E/S la encola e intenta unirla a otra petición de bloques adyacentes en la cola
- Además ordena las peticiones usando como criterio los sectores que acceden en disco

● ● ● | Ascensor de Linux (elevator)

- Diferentes estrategias para despachar las peticiones de E/S
- Básica: ascensor que no cambia de sentido hasta llegar a un extremo
- Noop (No Operation)
 - Sin reordenación
- CFQ (Complete Fairness Queueing)
 - RR entre diferentes procesos
 - Usualmente muchas colas (64) y se usa un hash sobre el PID para encolar en la cola correspondiente por orden de llegada las peticiones en las diferentes colas

● ● ● | Ascensor de Linux (elevator)

● Deadline

- Las misma peticiones encoladas por dos sistemas en cuatro colas
- 2 read/write ordenadas por sector
- 2 read/write ordenadas por deadline (500ms lecturas y 5 s escrituras)
- Se intentan primero las read (write si lleva mucho tiempo sin tocarle)
- Después se mira si hay algún deadline cumplido sino sigue por orden donde estaba
- Se despacha un lote, más grande si son contiguas

● ● ● | Ascensor de Linux (elevator)

- Anticipatory

- Igual que deadline (125 ms para read, 250 ms para write)
- Algunas veces cambia de sentido (cuando la distancia hacia atrás es menos de la mitad)
- Algunas veces espera un poco (7 ms) a un proceso
 - Que repite un patrón de peticiones (acumular estadísticas de los procesos)

● ● ● | Configurar el planificador

- Configurable como root en los ficheros:

/sys/block/{DEVICE-NAME}/queue/scheduler

```
# cat /sys/block/sda/queue/scheduler
```

```
noop anticipatory deadline [cfq]
```

```
# echo noop > /sys/block/hda/queue/scheduler
```

```
# cat /sys/block/sda/queue/scheduler
```

```
[noop] anticipatory deadline cfq
```

● ● ● | Lecturas

- Understanding the Linux Kernel:
 - Capítulo 15. “Page cache” (completo)
 - Capítulo 14
 - Block Devices Handling
 - The Generic Block Layer
 - The I/O Scheduler
- Linux Kernel Development:
 - Capítulo 16. The Page cache and ...
 - Capítulo 14. The Block I/O layer