BLOCK I/O LAYER

Dispositivos de bloques: Son dispositivos HW que se distingue por el acceso aleatorio a trozos de datos de tamaño fijo. Estos trozos de datos son los bloques (disco duro).

Dispositivos de caracteres: Se accede como un stream de datos secuenciales, un byte tras otro.

Manejar estos tipos de datos hay que tener en cuenta lo siguiente: los dispositivos de caracteres solo tienen una posición (como la tecla de un teclado) y los dispositivos de bloques son capaces de navegar hacia delante y atrás en cualquier ubicación en los datos.

- **ANATOMÍA DEL DISPOSITIVOS DE BLOQUES**

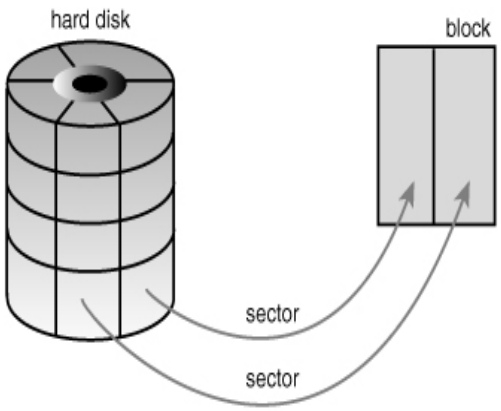
La unidad más pequeña en un bloque son los sectores, vienen en potencias de 2 y lo más común son 512 bytes. El tamaño del sector es una propiedad física del dispositivo y es la unidad fundamental en todos los dispositivos de bloques.

Muchos dispositivos de bloques pueden operar en múltiples sectores a la vez.

El bloque es una abstracción del sistema de ficheros, los sistemas de ficheros pueden ser accedidos en múltiplos de un bloque. Están compuestos por uno o más sectores y no es más grande que una página.

Aunque el dispositivo físico es direccionable a nivel de sector, el kernel realiza todas las operaciones de disco en términos de bloque. Como la unidad lógica de un dispositivo es el sector, el tamaño de bloque no puede ser más pequeño que el sector y tiene que ser múltiplo de este.

Conclusión: los sectores son la unidad direccionable más pequeña en un dispositivo y los bloques son la unidad más pequeña en un filesystem.



- **BUFFERS Y BUFFER HEAD**

Cuando un bloque es guardado en memoria, es almacenado en un buffer. Cada buffer está asociado a un bloque y representa un bloque de disco en la memoria.

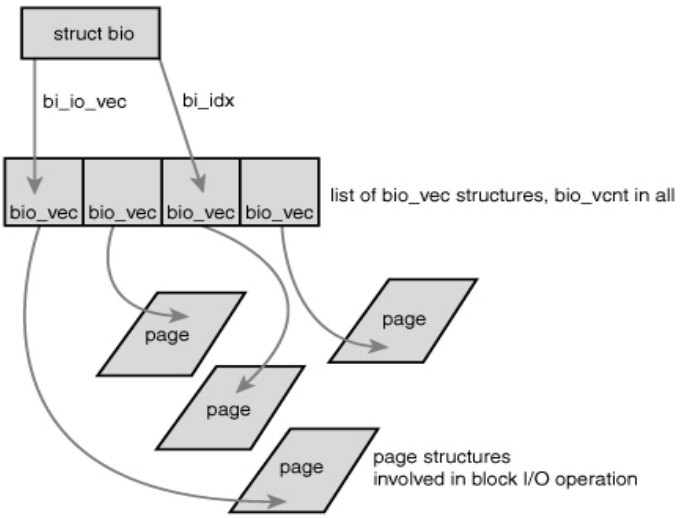
```
struct buffer_head {
        unsigned long b_state;       /* buffer state flags */
        struct buffer_head *b_this_page; /* list of page's buffers */
        struct page *b_page;         /* associated page */
        sector_t b_blocknr;          /* starting block number */
        size_t b_size;               /* size of mapping */
        char *b_data;                /* pointer to data within the page */
        struct block_device *b_bdev; /* associated block device */
        bh_end_io_t *b_end_io;       /* I/O completion */
        void *b_private;             /* reserved for b_end_io */
        struct list_head b_assoc_buffers;  /* associated mappings */
        struct address_space *b_assoc_map; /* associated address space */
        atomic_t b_count;            /* use count */
};
```

El propósito del buffer head es describir un mapeo entre el bloque en el disco y el buffer físico en memoria. Tiene dos problemas:

- Estructura de datos grande y poco manejable, ni limpia ni sencilla de manipular.
- Describen únicamente un buffer.

- **BIO STRUCTURE**

El contenedor básico para la E/S en bloque dentro del kernel es la bio structure. Esta estructura representa las operaciones de E/S en bloque que están activas como una lista de segmentos. Un segmento es un trozo de un búfer que es contiguo en la memoria. Por lo tanto, los búferes individuales no necesitan ser contiguos en la memoria. Al permitir que los búferes se describan en trozos, la estructura bio proporciona la capacidad para que el núcleo realice operaciones de E/S en bloque incluso de un solo búfer desde múltiples ubicaciones en la memoria.



- **VECTORES E/S**

El campo bi_io_vec apunta a una matriz de estructuras bio_vec. Estas estructuras se utilizan como listas de segmentos individuales en esta operación de E/S de bloque específica. Cada bio_vec se trata como un vector de la forma <página, offset, len>, que describe un segmento específico: la página física en la que se encuentra, la ubicación del bloque como un offset en la página, y la longitud del bloque a partir del offset dado.

```
struct bio_vec {
        /* pointer to the physical page on which this buffer resides */
        struct page     *bv_page;

        /* the length in bytes of this buffer */
        unsigned int    bv_len;

        /* the byte offset within the page where the buffer resides */
        unsigned int    bv_offset;
};
```

- **BUFFER HEADS VS BIO STRUCTURE**

La diferencia entre las buffer heads y la estructura bio es que la estructura bio representa una operación de E/S, que puede incluir una o más páginas en memoria. Por otro lado, la estructura buffer_head representa un único buffer, que describe un único bloque en el disco. Dado que las buffer heads están vinculadas a un único bloque de disco en una única página, provocan la división innecesaria de las peticiones en trozos del tamaño de un bloque, para luego volver a reunirlos. Como la estructura bio es ligera, puede describir bloques discontinuos y no divide innecesariamente las operaciones de E/S.

Mejoras de la bio structure respecto a buffer head:

- Representa más memoria, representa tanto E/S de página normal como la E/S directa, facilita la realización de operaciones de E/S en bloque dispersos y es una estructura más ligera.

- **COLAS DE PETICIONES**

Los dispositivos de bloque mantienen colas de peticiones para almacenar peticiones de E/S de bloque pendientes. La cola de peticiones contiene una lista doblemente enlazada de peticiones e información de control asociada. Mientras la cola de peticiones no esté vacía, el controlador del dispositivo de bloque asociado a la cola toma la petición de la cabeza de la cola y la envía a su dispositivo de bloque asociado. Cada elemento de la lista de peticiones de la cola es una única petición. Cada una de las peticiones puede estar compuesta por más de una bio structure.

- **PLANIFICADORES E/S**

Enviar simplemente las peticiones a los dispositivos de bloque en el orden en que el kernel las emite, tan pronto como las emite, da como resultado un rendimiento pobre. Una de las operaciones más lentas en un ordenador moderno es la búsqueda en el disco.

Para minimizar esta operación de utilizan los planificadores E/S.

Un planificador E/S trabaja gestionando la cola de peticiones de un dispositivo de bloque. Decide el orden de las solicitudes en la cola y en qué momento se envía cada solicitud al dispositivo de bloque. Gestiona la cola de peticiones con el objetivo de reducir las búsquedas, lo que aumenta el throughput. Realizan dos operaciones que son merging y sorting.

- **LINUX ELEVATOR**

Cuando se añade una solicitud a la cola, primero se comprueba con todas las demás solicitudes pendientes, si hay una solicitud a un sector adyacente en el disco en la cola, la solicitud existente y la nueva se fusionan en una sola solicitud. Hay dos tipos de merging:

- Front merged, si la nueva solicitud es inmediatamente posterior a una solicitud existente.
- Back merged, si la nueva solicitud es inmediatamente anterior a una solicitud existente.

Si una solicitud en la cola es lo suficientemente antigua, la nueva solicitud se inserta en la cola de la cola para evitar que las otras solicitudes más antiguas se queden sin respuesta.

Si hay una ubicación adecuada por sectores en la cola, la nueva solicitud se inserta allí. Esto mantiene la cola ordenada por ubicación física en el disco.

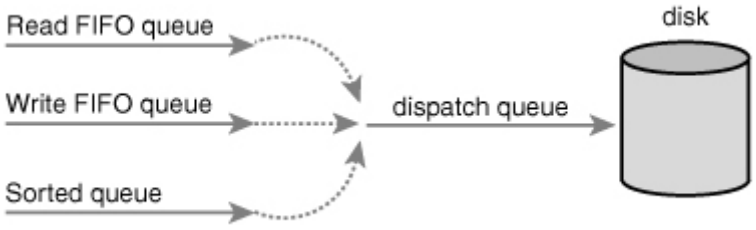
Por último, si no existe un punto de inserción adecuado, la solicitud se inserta en la cola de la cola.

- **DEADLINE I/O SCHEDULER**

Busca evitar la inanición causada por el elevador Linux debido a que las operaciones de E/S pesadas en un área del disco pueden hacer que las operaciones de solicitud en otra parte del disco se queden indefinidamente sin atender. Trabaja de la misma manera que el anterior al insertar peticiones, sin embargo, también inserta la solicitud en una segunda cola que depende del tipo de solicitud. Las solicitudes de lectura se ordenan en una cola especial de lectura FIFO, y las solicitudes de escritura se insertan en una cola especial de escritura FIFO.

En este planificador cada solicitud está asociada a un tiempo de caducidad. Si la solicitud en la cabeza de la cola FIFO de escritura o de lectura expira, se comienza a atender las solicitudes de la cola FIFO. De esta manera, se intenta asegurar que ninguna solicitud quede pendiente más allá de su tiempo de expiración.

Debido a que las solicitudes de lectura tienen un valor de expiración sustancialmente menor que las solicitudes de escritura, el planificador trabaja para asegurar que las solicitudes de escritura no se queden sin respuesta. Esta preferencia hacia las solicitudes de lectura proporciona una latencia de lectura minimizada.



- **ANTICIPATORY I/O SCHEDULER**

Pretende seguir proporcionando una excelente latencia de lectura, pero también un excelente rendimiento global. Implementa tres colas (más la cola de envío) y vencimientos para cada solicitud. El principal cambio es la adición de una heurística de anticipación.

Este planificador intenta minimizar la latencia búsqueda que acompaña a las peticiones de lectura emitidas durante otra actividad de E/S del disco. Cuando se emite una solicitud de lectura, se maneja como de costumbre. Sin embargo, después de que se envíe la solicitud, no busca inmediatamente y vuelve a gestionar otras solicitudes. En su lugar, no hace absolutamente nada durante unos milisegundos. Una vez transcurrido el periodo de espera, el planificador busca de nuevo donde lo dejó y continúa gestionando las peticiones anteriores.

Si no se hace nada cuando se espera esto podría perder en rendimiento, pero para ello, se implementa una heurística para anticipar correctamente. Esto resulta en baja latencia y alto rendimiento.

Funciona bien en la mayoría de las cargas de trabajo. Es ideal para los servidores

- **COMPLETE FAIR QUEUING I/O SCHEDULER**

Asigna las solicitudes de E/S entrantes a colas específicas basadas en el proceso que origina la solicitud de E/S. Por ejemplo, las peticiones de E/S del proceso foo van en las colas de foo. Dentro de cada cola, las peticiones se unen a las adyacentes y se ordenan por inserción. Las colas se mantienen así ordenadas por sectores.

Atiende las colas de forma rotatoria, extrayendo un número configurable de peticiones de cada cola antes de continuar con la siguiente. Esto proporciona equidad a nivel de proceso, asegurando que cada proceso recibe una porción justa del ancho de banda del disco.

Recomendado para desktop workloads.

- **NOOP I/O SCHEDULER**

No realiza sorting ni búsquedas, pero sí realiza la fusión, como única tarea. Cuando se envía una nueva solicitud a la cola, se fusiona con cualquier solicitud adyacente. Mantiene la cola en un orden casi FIFO del que el controlador del dispositivo de bloque puede arrancar peticiones.

Si un dispositivo de bloque tiene poca o ninguna sobrecarga asociada a la "búsqueda", entonces no hay necesidad de ordenar las solicitudes entrantes, por lo que resulta útil. Se utiliza en dispositivos de acceso aleatorio.

- **FINAL**

Para elegir uno, elevator = foo; Estos son los diferentes parámetros:

Parameter	I/O Scheduler
`as`	Anticipatory
`cfq`	Complete Fair Queuing
`deadline`	Deadline
`noop`	Noop