Contenido

Contenido

Presentación

La primera descripción del sistema operativo UNIX data del año 1969, y corresponde a la formulación de Thomson y Dennis Ritchie, trabajadores entonces de la compañía americana AT&T. En 1973 apareció la primera versión de UNIX totalmente programada en C.

El sistema operativo UNIX era en principio gratuito pero, en 1982, Bell Laboratories lanzó al mercado la versión comercial UNIX System III. Poco después aparece el UNIX System IV, que representa la base del actual sistema standard de UNIX, que es el UNIX System V. Existen varias distribuciones del UNIX System V:

- UNIX SVR1, que incorpora el editor VI y la primera edición de la librería Curses;
- UNIX SVR2 presenta el sistema de archivo;
- SVR2.1 incluye la paginación bajo demanda;
- SVR3 sale al mercado en 1987, y presenta como novedad más destacable el manejo de redes.
- Por último, el UNIX SVR4 unifica los criterios de implementación de los sistemas anteriores.

Existen otras variantes de UNIX, entre las que destaca el UNIX BSD (Berkeley Software Distribution), diseñado en Berkeley, que incluye el editor de comandos C−Shell, algunos editores de texto nuevos y un compilador de Pascal.

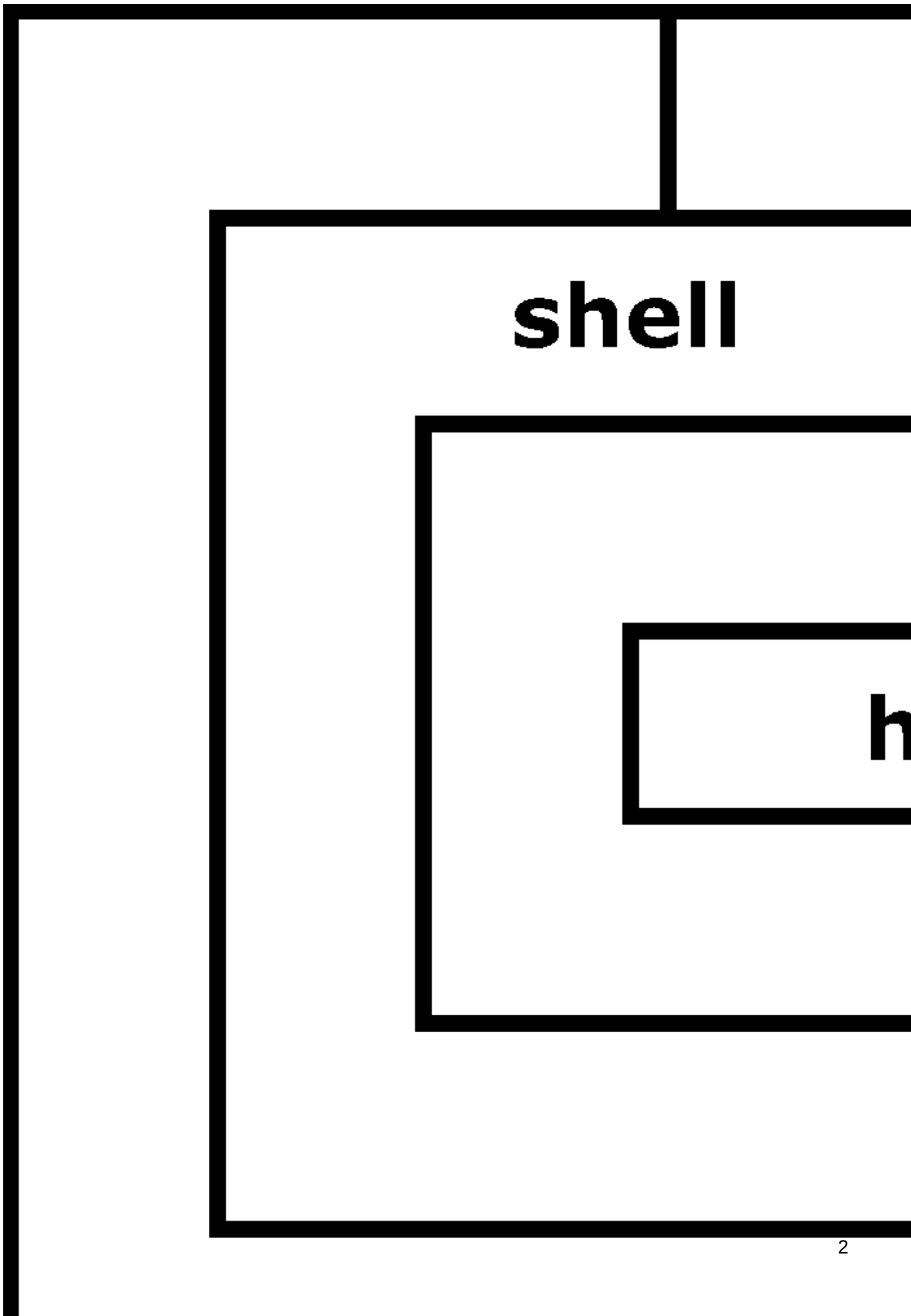
Jay y Halley Pusictes incorporaron al sistema UNIX soporte para redes en el año 1984. Jay es fundador de Sun Microsystems, compañía que añade a los estándares UNIX el NFS (Network File System).

Otros sistemas UNIX son el UnixWare (Novell), SunOS y Solaris (Sun MicroSystems), Aix (IBM), Digital UNIX (Digital), HP−UX (Hewlett Packard), Xenix (Microsoft), SCO−UNIX (Santa Cruz Operations) y, por último, Linux, versión de UNIX de libre distribución creada gracias al esfuerzo de multitud d usuarios y programadores sin ánimo de lucro.

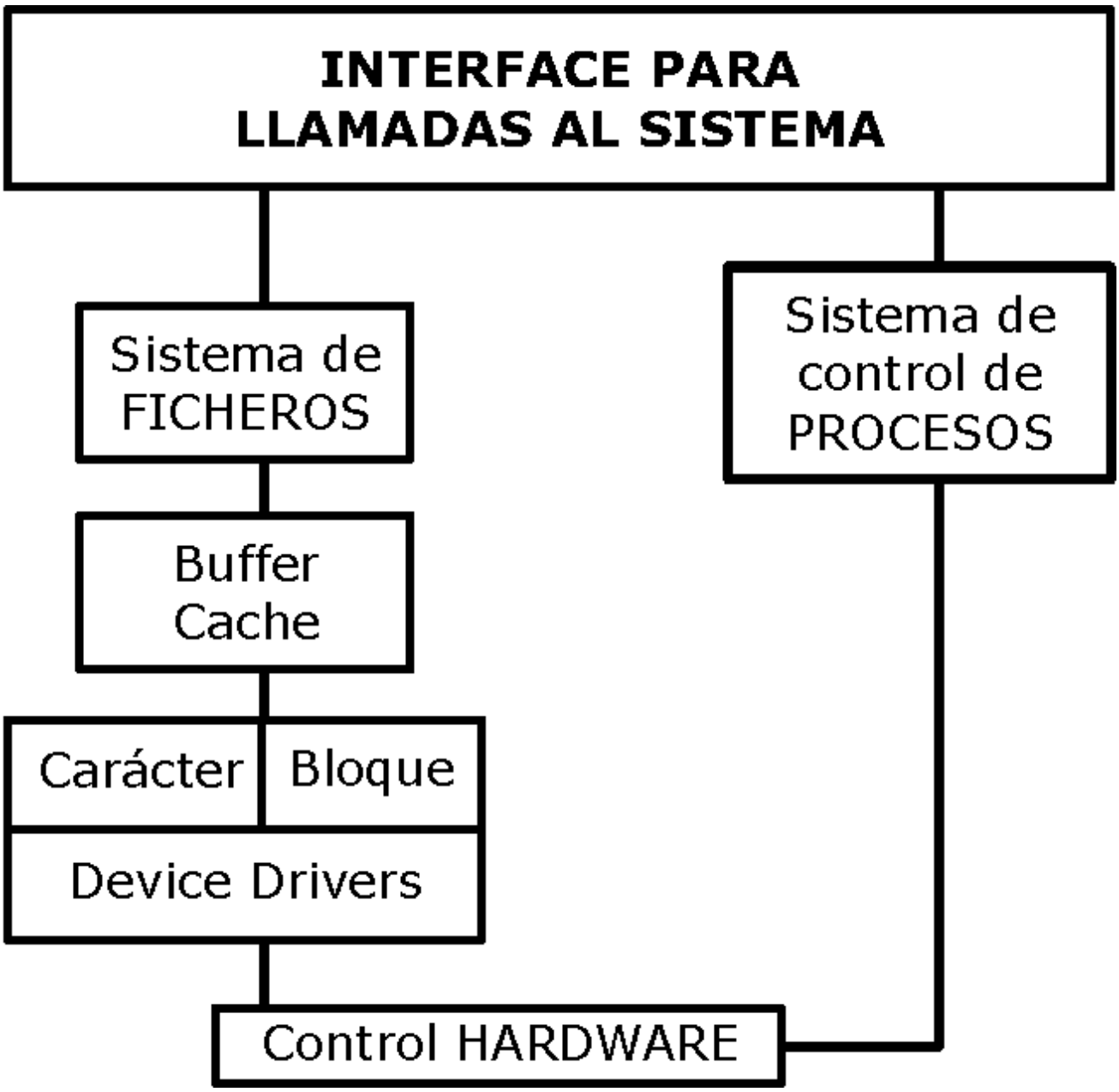
Arquitectura general de UNIX

El sistema UNIX está basado en tres niveles:

- Nivel USUARIO
- Nivel KERNEL
- Nivel HARDWARE


shell
h

Nivel KERNEL



El kernel tiene una interfaz con los programas de usuario, teniendo las llamadas al sistema apariencia de funciones C, distinguiéndose dos tipos:

- Llamadas al sistema de ficheros
- Llamadas al control de procesos

Funciones del kernel

El kernel:

- Controla la ejecución de procesos;
- Planifica la CPU en tiempo compartido;
- Asigna memoria a los procesos;
- Asigna memoria en disco;

- Permite a los procesos el acceso a dispositivos.

La ejecución de un proceso en UNIX se divide en dos niveles: nivel usuario y nivel kernel. Cuando se produce una llamada al sistema se pasa del modo usuario al modo kernel. Éste analiza la llamada, la ejecuta y devuelve el control a modo usuario. Esta diferenciación de modo se produce porque los procesos en modo usuario pueden acceder a sus instrucciones y datos, pero no a instrucciones y datos del kernel o de otros usuarios; mientras que el modo kernel puede acceder a todos los datos e instrucciones del sistema. Hay instrucciones privilegiadas a las que sólo se puede acceder en modo kernel, el cual reside permanentemente en memoria.

El buffer cache

Introducción

El buffer cache consiste en un conjunto de buffers internos de datos manejados por el kernel con el objetivo de minimizar la frecuencia de acceso a disco.

Estos buffers contienen bloques de datos de disco utilizados recientemente. Cuando el kernel quiere leer el disco, se comprueba primeramente si los datos requeridos están en el buffer cache: si están, no es necesario el acceso a disco; si no es así no hay más remedio que acudir al disco.

Si la operación a realizar es de escritura en disco, el kernel no la realiza directamente, sino que lo hace sobre el buffer, quedando los datos almacenados allí para posteriores lecturas.

El espacio ocupado por el buffer cache es configurable en la inicialización del sistema, reservándose un cierto número de buffers. Cada buffer contiene los datos que se corresponden con un bloque de disco: es una copia en memoria del bloque de disco. Asimismo, ningún bloque puede estar en más de un buffer; es decir, un bloque de disco no podrá tener más de un bloque de memoria.

Estructura

El buffer cache está formado por una serie de buffers organizados. Hay una lista de buffers libres (`*free list*'), y una serie de colas (*hash*) para facilitar el acceso a los buffers sin recorrerlos todos.

Cada buffer está constituido por una cabecera y un área de datos, que es un array de memoria donde se almacenan los datos de disco contenidos en el buffer.

La distribución del buffer queda como sigue:

dispositivo
bloque
Status
Puntero al área de datos
Puntero siguiente en la cola Hash
Puntero anterior en la cola Hash
Puntero siguiente en la *free list*
Puntero anterior en la *free list*

- Los dos primeros campos (número de **dispositivo** y de **bloque**) se utilizan como identificativo del buffer.
- El campo **status** incluye varios campos que indican el estado actual del buffer:

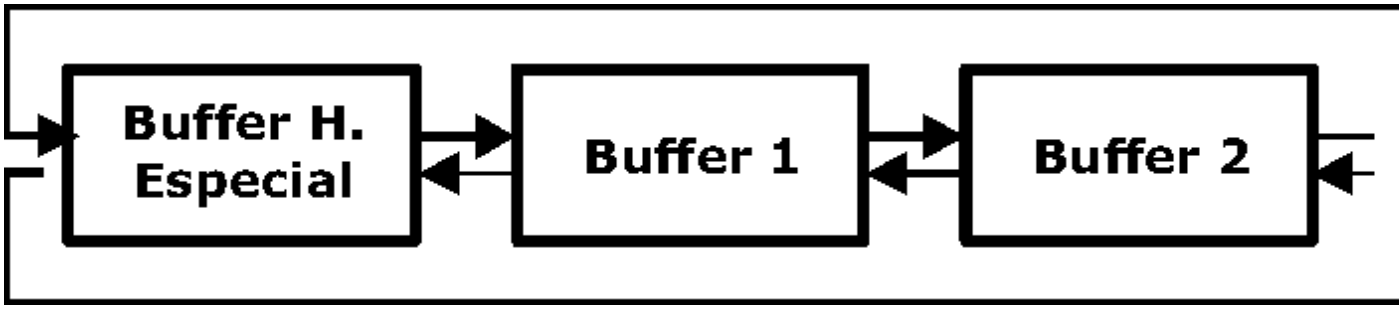
- Si está libre u ocupado (utilizado por un proceso);
- Si los contenidos son válidos o no,
- Si el buffer es de escritura retardada;
- Si se produce una lectura o escritura;
- Si hay procesos en espera para utilizar el buffer.

La asignación de estos campos se hace con un LRU de bloques de disco al buffer.

- El **primer puntero** señala el **área de datos**, que es la zona de memoria donde están los datos del bloque del disco.
- Los punteros 2º y 3º apuntan al buffer posterior y anterior en la *cola hash*.
- Los punteros 4º y 5º indican los buffers posterior y anterior en la *free list*.

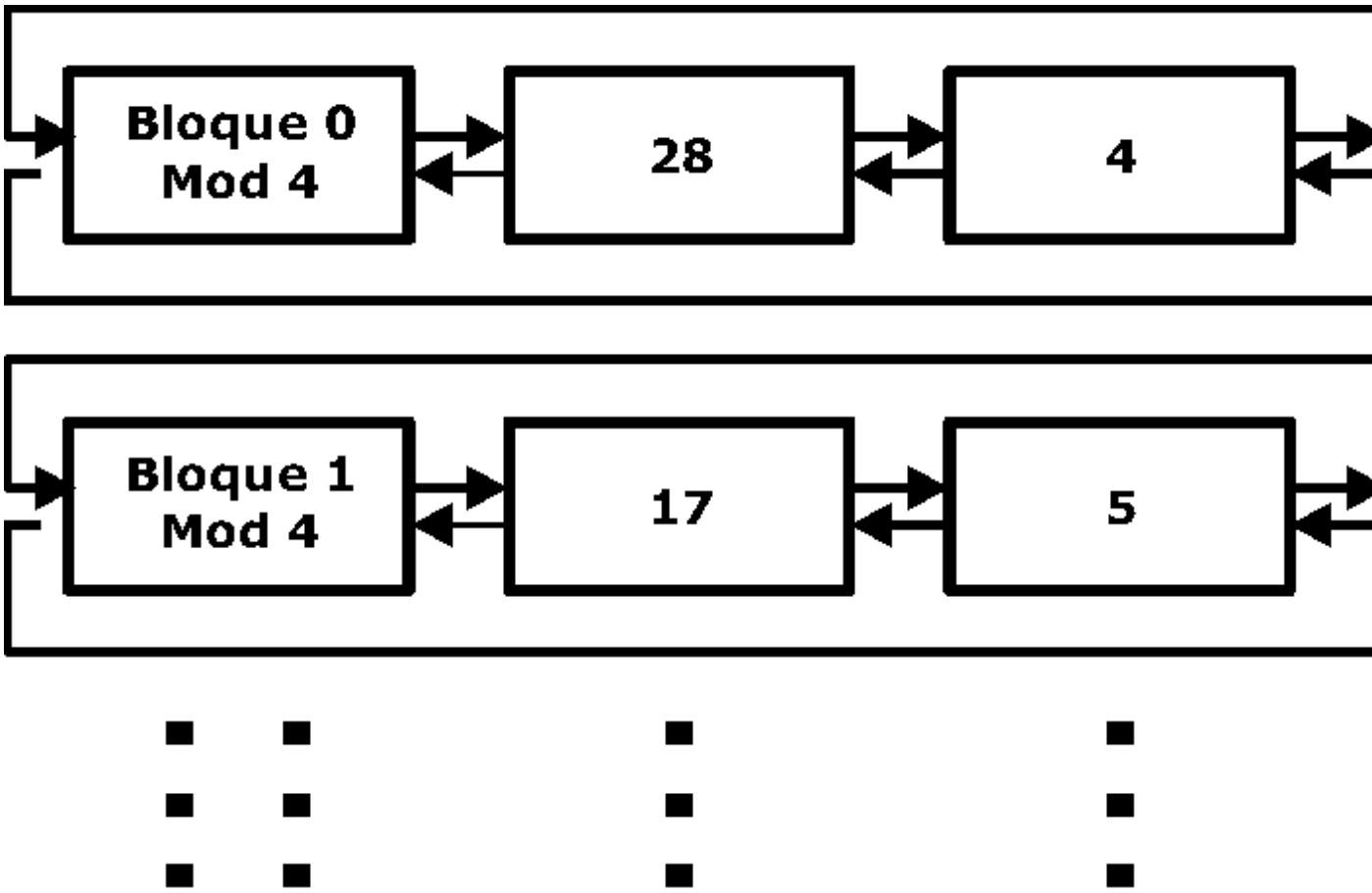
En el buffer cache se distinguen dos estructuras de datos:

- **Free List** (única): es una lista circular doblemente enlazada, siendo los elementos de esta lista buffer headers. Tiene un buffer especial al principio de la lista, usado para marcar el principio y el final de la lista.



Mediante el uso de la lista con un LRU se realiza la asignación de los buffers a los bloques. Cuando se asigna un buffer a un bloque de disco no se puede volver a asignar sin que se hayan asignado todos los componentes de la lista antes. Si se realiza la asignación de un buffer de disco se asignará aquel que encabece la lista. Si se libera un buffer asignado a bloque, el buffer liberado se colocará al final de la free list. Con este sistema se mantiene el algoritmo LRU automáticamente.

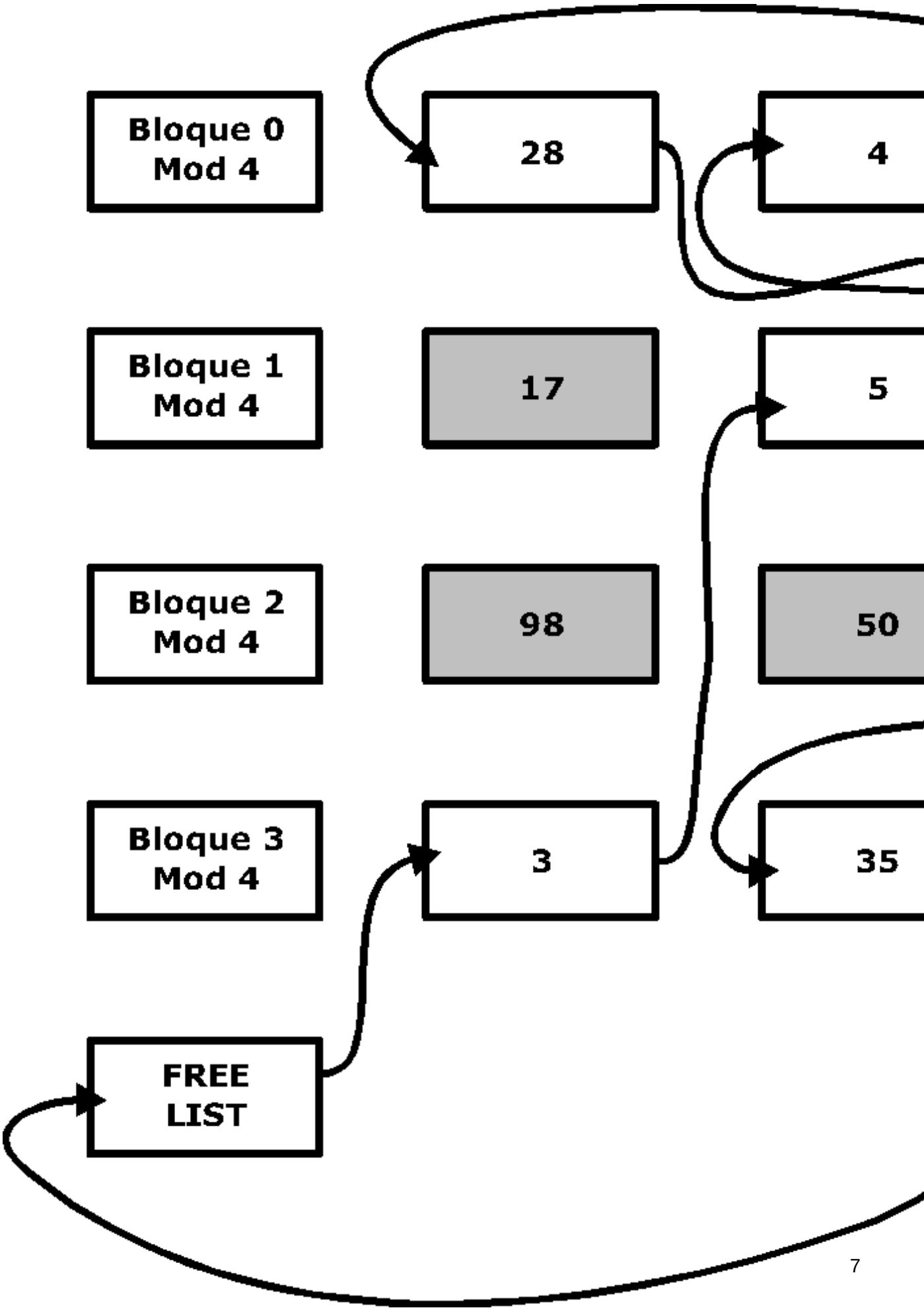
- **Colas Hash** (múltiples). Son una serie de colas circulares doblemente enlazadas, con un buffer especial que indica principio y final de la cola. La diferencia con la free list radica en que la cola hash está organizada en una tabla hash según el número de bloque y el número de dispositivo lógico, siendo así más rápida la búsqueda.



Los datos que se corresponden con los datos de disco sólo pueden estar en una de las colas hash.

Un buffer puede estar a la vez en una de las colas hash y en la free list. En caso de que este buffer no se encuentre ocupado por un proceso, la lista de bloques se organiza a partir de la estructura de colas hash.

Suponiendo que los buffers 64, 17, 98 y 50 están ocupados, la free list quedaría como sigue:


Bloque 0
Mod 4
28
4
Bloque 1
Mod 4
17
5
Bloque 2
Mod 4
98
50
Bloque 3
Mod 4
3
35
FREE
LIST

Funcionamiento

Son necesarios cuatro algoritmos.

1/ Asignación de un bloque de disco a un buffer (Get Block):

Recibe como entrada el número de dispositivo y de bloque.

- Encuentra el bloque y está libre: se marca como ocupado, se quita de la free lista. Ya está en su cola hash.
- No se encuentra en su cola hash: se asigna el primer bloque de la free list, se marca como ocupado y se coloca en la cola correspondiente.
- No se encuentra en la cola hash y se le asigna uno de escritura retardada: lo marca como ocupado, realiza una escritura asíncrona en disco e inicia la búsqueda de un nuevo bloque.
- No se encuentra en su cola hash y la lista de bloques libres está vacía: el proceso ha de esperar y se va a dormir. Cuando el algoritmo de liberación actúa despierta a los procesos dormidos.
- El bloque está ocupado: el proceso ha de esperar hasta que se libere.

2/ Algoritmo de liberación de buffer:

Libera el buffer, colocándolo como libre y situándolo al final de la lista de libres, no volviendo a ser utilizado hasta que sean utilizados todos los demás integrantes de la free list que lo preceden.

Se ha de avisar a los procesos en espera de que se libera ese buffer, o de que otro buffer ha sido liberado.

3/ Algoritmo de lectura de un bloque:

- Busca el buffer en el buffer cache por medio del algoritmo obtener bloque, que marcará el buffer como ocupado.
- Si está en el buffer cache, se devuelve el contenido, no se precisa la lectura en disco.
- En caso contrario, se llama al driver del disco para que solicite una lectura.
- El driver calcula la dirección y llama al controlador del disco.
- Cuando se termina la lectura, el controlador provocará una interrupción y el servicio de interrupciones avisa a los procesos en espera de que ese buffer ha quedado libre.
- Libera el buffer.

4/ Algoritmo de escritura en disco

Es similar a la lectura de un bloque. Existen dos tipos de escritura:

- Síncrona: el proceso realiza la operación de forma síncrona, esto es, espera a que se termine la operación de escritura en disco para liberar el buffer.
- Asíncrona: el proceso no espera a que se termine la escritura. Cuando se termina la E/S se produce una interrupción, y es el servicio de interrupciones el que se encarga de liberar el buffer.

Ventajas y desventajas del buffer cache

Ventajas:

- El acceso a disco es más uniforme, pues se hace siempre a través del buffer cache. Esto provoca un código más modular.
- Aumenta la velocidad del disco.
- Aumenta la integridad, ya que un bloque no puede estar en dos buffers.

Desventajas:

- Sistema sensible a cortes de electricidad.
- Para grandes cantidades de datos, puede hacer lento el acceso a disco.

Sistema de ficheros

Para analizar el sistema de ficheros, debemos tener en cuenta dos puntos de vista: el del usuario y el del kernel.

Usuario

- Ve una estructura jerárquica que permite crear, borrar y modificar ficheros y directorios.
- Permite un crecimiento dinámico de los ficheros.
- Protege los datos de los ficheros.
- Trata los periféricos como ficheros: los permisos de lectura o escritura en un periférico son iguales a los de cualquier otro fichero.
- Tanto los ficheros como los directorios son una sucesión de bytes.
- Cada uno de ellos está representado por un i−node, que tiene información acerca del fichero o directorio. Cada fichero tiene un único i−node, aunque puede tener varios nombres. Cada uno de esos nombres es un link.
- El UNIX maneja tres estructuras: tabla de i−nodes, tabla de ficheros y tabla de escritores de ficheros de usuario.

Esta última tabla tiene una entrada para cada usuario que abre un fichero. Además tiene información de otras tablas, como son una tabla particular para cada proceso (tabla de ficheros), e información de otras tablas (una por sistema de ficheros). Esta tabla contiene el descriptor de fichero, que es el número que se emplea como índice para acceder a l tabla de descriptores de fichero, y un puntero a donde está el fichero en la tabla de ficheros.

I−nodes

La implementación de ficheros utiliza una técnica de índices con múltiples niveles. Cada fichero tiene asociado un índice llamado **i−node** (**i**ndex **node**).

En disco, un sistema de ficheros UNIX tiene el siguiente aspecto:

Boot Block	**Super Block**	**I−node List Blocks**

- **Boot Block:** Contiene el código para inicializar el UNIX. Todo sistema de ficheros tiene boot block, aunque no sea estrictamente necesario.
- **Super Block:** Indica el estado y configuración general del bloque de fichero: tamaño, bloques libres
- **I−node list blocks:** Serie de bloques donde están de forma contigua en el disco los i−nodes de todos los ficheros. Cuando se habla de i−node list nos referimos a un array en disco donde cada elemento es un i−node de un fichero.
- **Data blocks:** Bloques de datos de los ficheros.

Conviene diferenciar la unidad física o disco, que puede tener uno o más sistemas de ficheros, cada uno con esta misma estructura.

Cada sistema de ficheros se denomina dispositivo lógico, mientras que el disco es el dispositivo de almacenamiento físico. El kernel es el encargado de asignar un número a cada dispositivo lógico o sistema de ficheros. El encargado de traducir las direcciones del dispositivo lógico al físico es el controlador de E/S (device driver).

Campos por i−node en la lista de i−nodes del disco

- **Propietario** identifica al propietario del fichero asociado a ese i−node, distinguiendo entre individual y el grupo.
- En el campo **Permisos** se expresan las posibilidades de operación según sea el propietario, un usuario de su mismo grupo o un usuario cualquiera del sistema quien intente la operación. Los permisos son de lectura (**R**ead), escritura (**W**rite) y ejecución (e**X**ecute).
- Tipo de fichero indica información sobre el tipo especial de fichero, siendo algunos ejemplos:

- **(−)**: fichero regular;
- **(d)**: directorio;
- **(b)**: dispositivo de bloque;
- **(c)**: dispositivo de carácter;
- **(ø)**: i−node libre.

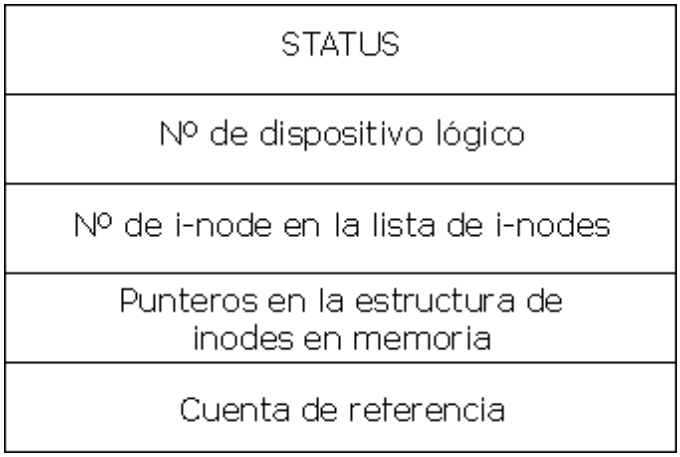
- El campo **Tiempos de fichero** tiene varios subcampos:

- Última modificación del fichero;
- Último acceso al fichero;

- Última modificación del i−node causada, además de por un cambio en los datos del fichero, por un cambio en los propietarios, permisos o links, sin necesidad de cambiar los datos del fichero.

- El campo **Número de links** expresa el número de nombres del fichero dentro de la jerarquía.
- **Tamaño del fichero** muestra el tamaño del fichero en Bytes.
- El campo **Tabla de contenidos** almacena punteros a diversos bloques de datos. Por ejemplo, en UNIX System V la tabla de contenidos consta de 10 punteros directos a bloques de datos, un puntero indirecto, otro doble y otro triple. El número de punteros directos no es standard.

El kernel mantiene en memoria una tabla de i−nodes con la siguiente información:



- El campo **Status** indica posibles estados del i−node:

- Si el i−node está ocupado;
- Si hay un proceso en espera de que el i−node quede libre;
- Si el contenido del i−node en memoria es distinto del i−node del disco;
- Si el contenido del fichero en el disco es distinto del contenido del fichero en memoria;
- Si el i−node es un punto de montaje; es decir, si del i−node se haya `colgado' otro sistema de ficheros.

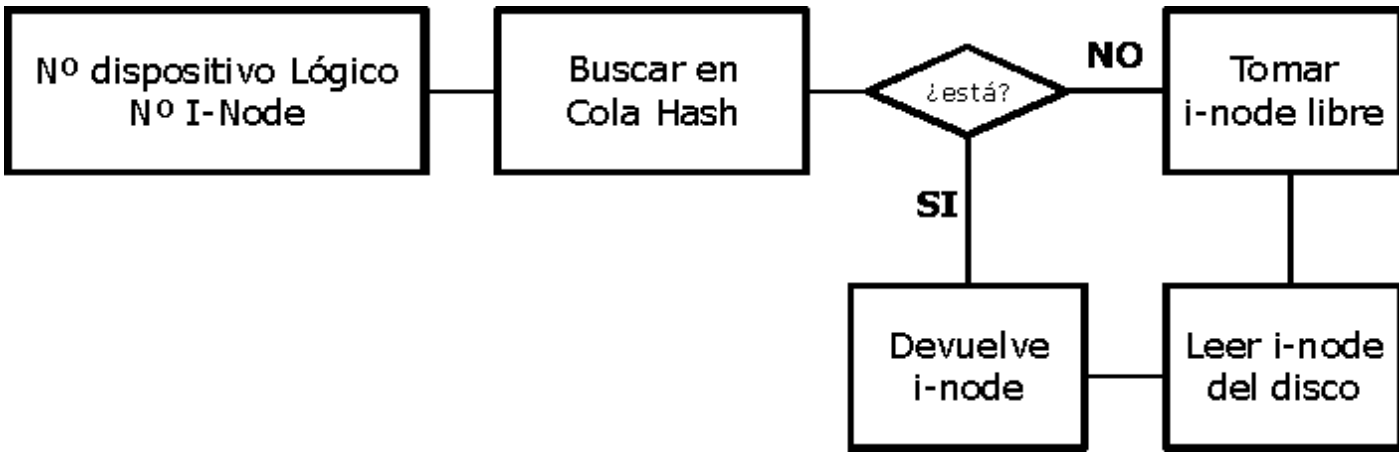
- El cuarto campo contiene los punteros necesarios para mantener la estructura de los i−nodes en memoria, que es similar a la del buffer cache: existen una serie de colas hash identificadas por el número del i−node y el número de dispositivo lógico, junto con una lista de i−nodes libres.
- **Cuenta de referencia** indica el número de ficheros abiertos para ese i−node; es decir, el número de copias abiertas mediante una llamada de tipo *`open'* que están en unos, no cerradas por una llamada *`close'*.

Un i−node estará en la lista de libres si el indicador de cuenta de referencia está a cero. Cuando se abre un fichero, si el fichero está activo se incrementa la cuenta; si se cierra, ésta se decrementa, y si vale 0 se pasa el i−node a la lista de libres.

Algoritmos para la gestión de i−nodes

Mecanismo general de operación con i−nodes

Cuando se requiere el acceso a los datos de un fichero, primero se determina el número de dispositivo lógico y el número de i−node, a continuación se busca el i−node en la tabla hash correspondiente. Si está en la tabla se devuelve el i−node, si no es así se toma un i−node libre y luego se lee el i−node del disco; es decir, se pasa de la lista de i−nodes a la tabla de i−nodes.



I−Get

Toma un i−node de la tabla de i−nodes.

- Se parte del número de dispositivo lógico y del número del i−node.
- Si está en la cola hash es directo, sólo ha de devolver el i−node.
- Si no está en la cola hash, opera como sigue:

    - Toma el primer i−node de la lista de libres.
    - Lo marca como ocupado, ya que hay un proceso que accede al i−node.
    - Pasa los datos del i−node de disco (lista de i−nodes) al i−node en memoria (tabla de i−nodes).
    - Calcula, a partir del número de i−node en la tabla, el número de bloque en que se encuentra l i−node, aplicando la fórmula:

Nº BLOQUE = ((Nº Inode − 1) **DIV** (Nº inodes por bloque)) +
Nº del bloque del primer bloque de la lista de inodes

Por ejemplo: Nº del 1er bloque de la lista = 2; 8 inodes/bloque

Inode 8: ((8 − 1) DIV 8) + 2 = 2

Inode 9: ((9 − 1) DIV 8) + 2 = 3

- A partir del número de dispositivo y del número de bloque lee el bloque.
- Lee el contenido del i−node en el bloque. Para calcular la posición del i−node en el bloque se calcula el offset dentro de ese bloque de la siguiente forma:

Offset (inode) = ((Nº Inode − 1) **MOD** (Nº inodes por bloque)) %
% Tamaño (Bytes) del inode en disco

Siguiendo con el ejemplo anterior: Bloque 512 Bytes

512 / 8 64 Bytes / Inode

Offset 8: ((8 − 1) MOD 8) % 64 = 448 Bytes

- Retiene el i−node de la lista de libres.
- Copia el contenido en la cola hash correspondiente.
- Sitúa el i−node en la cola hash correspondiente.

- Incrementa la cuenta de referencia.
- El kernel le quita la marca de ocupado; es decir, lo deja libre, pero no pasa a la lista de libres porque su cuenta de referencia es distinta de 0.

En el caso de que la lista de libres esté vacía, se provoca un error si se requiere un i−node y hay demasiados ficheros abiertos.

I−put

Libera un i−node de la tabla de i−nodes.

- Decrementa la cuenta de referencia debido a una llamada *close*.
- Si la cuenta de referencia es Ø, entonces pasa a disco el i−node si es necesario; esto es, si ha sido modificado.
- Se pasa el i−node a la lista de libres.
- En el caso de que el número de links (nombres de ese fichero en el sistema de ficheros) sea Ø se libera todo el espacio; es decir, se liberan los bloques de disco asociados a ese fichero y se libera el i−node en disco.

Estructura y acceso a los ficheros regulares

En un i−node se encuentran las direcciones de bloque de disco que ocupa el fichero al que pertenece el i−node. En UNIX System V estas direcciones son 13 (punteros).

Según la tabla de contenidos, y dependiendo del tamaño de los bloques, variará el posible tamaño de los ficheros a manejar.

Por ejemplo, con bloques de 1K resultan los siguientes tamaños de ficheros, según se utilicen los índices directos, indirectos, indirectos dobles o indirectos triples:

Directos 10 K

Indirectos 256 K

Indirectos dobles 644 MB

Indirectos triples 166 GB

Ficheros directorio

Son, al igual que en MS−DOS, ficheros con información para construir la estructura jerárquica del sistema de ficheros. Esta estructura consiste en un array de entradas de tamaño constante de tal forma que cada entrada tiene la siguiente forma:

En UNIX System V, el tamaño del campo Nº I−node es de 2 Bytes, y el tamaño del nombre es de 14 Bytes. Los directorios son tratados por los procesos igual que los ficheros, con la salvedad de que cuando se crea una entrada de este tipo la realiza el kernel.

Permisos de acceso

- **Lectura (R)**: Permite leer el contenido del fichero directorio;
- **Escritura (W)**: permite escribir en el directorio, lo que supone crear y borrar archivos del directorio.
- **Ejecución (X)**: Permite la búsqueda de ficheros dentro del directorio (entrar en el directorio).

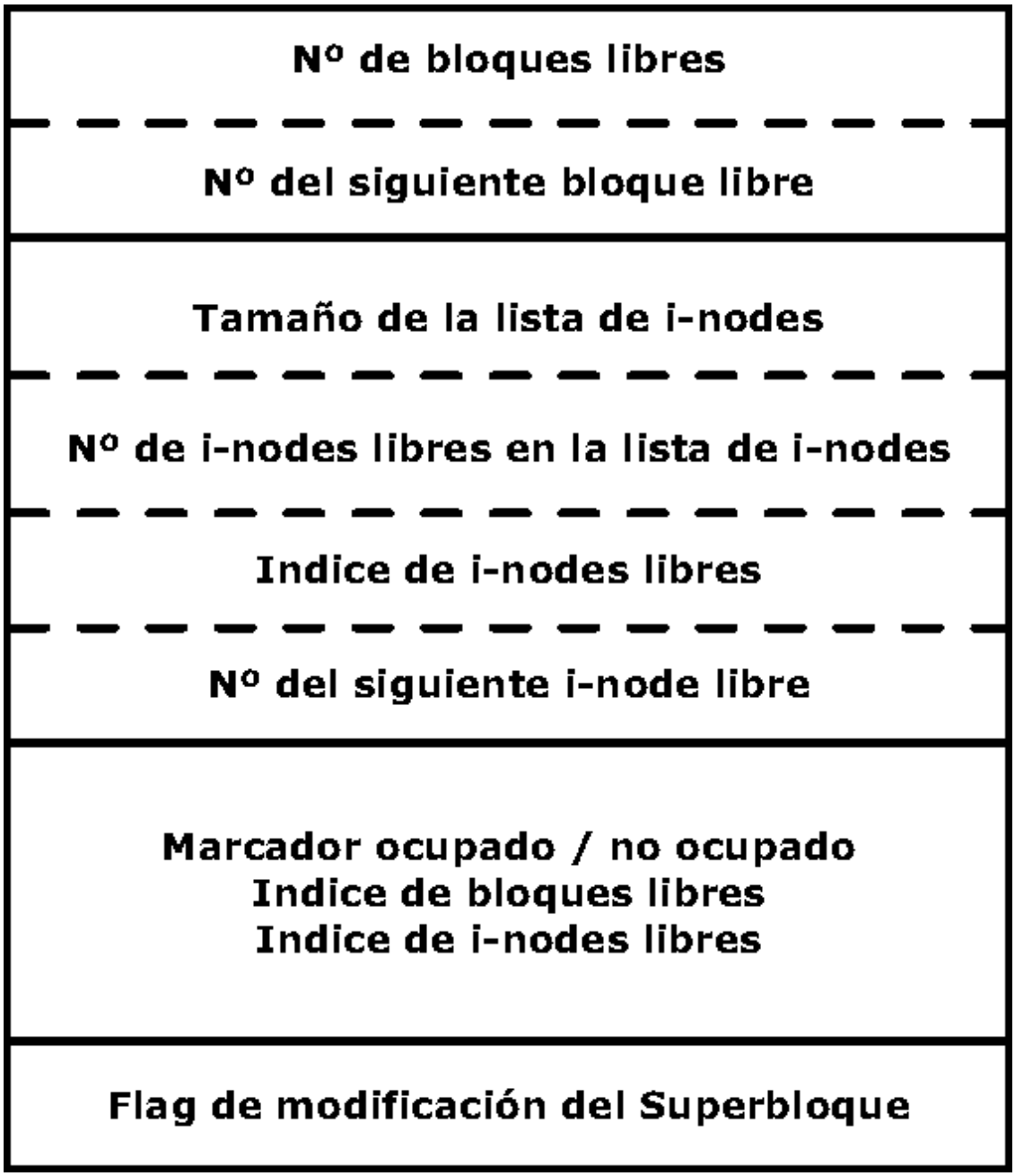
Al igual que en MS−DOS, existen las entradas y en cada directorio:

- contiene el número de i−node de ese directorio.
- contiene el número de i−node del directorio padre.

El directorio raíz contiene las dos entradas anteriores, y el número de i−node es el mismo

Super−Bloque

En el super−boque se halla almacenada la configuración general del sistema. Contiene la siguiente información:



Lista de bloques libres: No es una lista completa, sino que sólo contiene alguno de los bloques libres, que se encadenan con el resto de bloques libres del disco. El número de i−nodes libres se refiere al número de i−nodes no asignados a ningún fichero dentro de la lista de i−nodes. En la lista de i−nodes libres sólo están algunos de los i−nodes libres de la lista de i−nodes.

Ocupado / No ocupado: Contiene dos flags indicadores del estado de las listas y bloques de i−nodes. Los estados dependen de si están ocupados por un proceso o no.

El último flag expresa si el super−bloque ha sido modificado. Su presencia viene dada por la existencia de una doble copia del super−bloque: una en disco y otra en memoria. El flag indica la necesidad de actualizar la copia del super−bloque en disco si éste se modifica.