minix2.0

INODE

José Garrido Santana

Manuel Suárez Trujillo

tabla llamada i-node (nodo índice), la cual muestra:

- los atributos del fichero
- y las direcciones de sus bloques en disco.

Cuando el fichero es pequeño, se pueden almacenar en el mismo i-node las direcciones de los bloques de disco que ocupa, esto es, los **direcciona directamente**.

Para ficheros más extensos hay una entrada que apunta a un bloque de disco que contiene un conjunto de direcciones de bloque, es decir, los **direcciona indirectamente**.

Incluso puede haber una entrada de la tabla que apunte a un bloque que a su vez contiene entradas que apuntan a bloques para ficheros aún mayores, es decir, una **doble indirección**.

Existe un i-node para cada archivo de un disco en Minix. La dirección de este i-node se encuentra en la entrada del directorio para este archivo. La estructura de una entrada de directorio será, por tanto, de la siguiente forma :

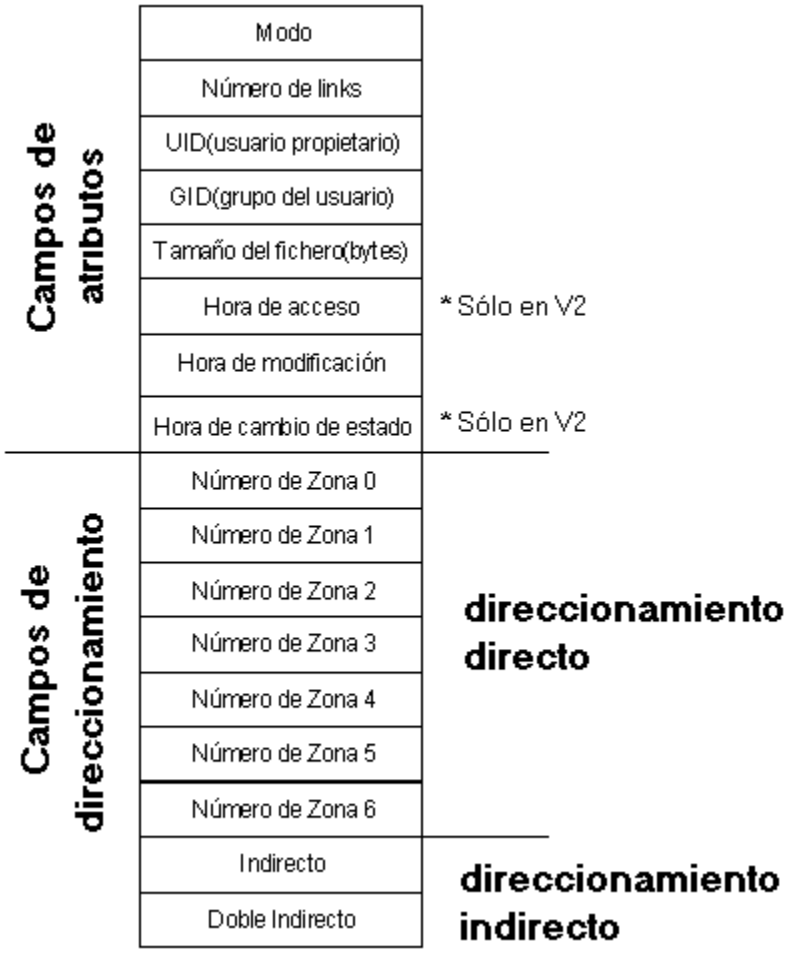
2 bytes	14 bytes
nº de i-node	Nombre del fichero

Cuando se abre un fichero, su i-node se localiza y se trae a la tabla de i-nodes de la memoria (*función get_inode*), donde permanece hasta que se cierra el fichero. La tabla de i-nodes tiene campos adicionales que no se encuentran en el disco como son el dispositivo del i-node y su número, de manera que el sistema sabe donde escribirlo si se modifica.

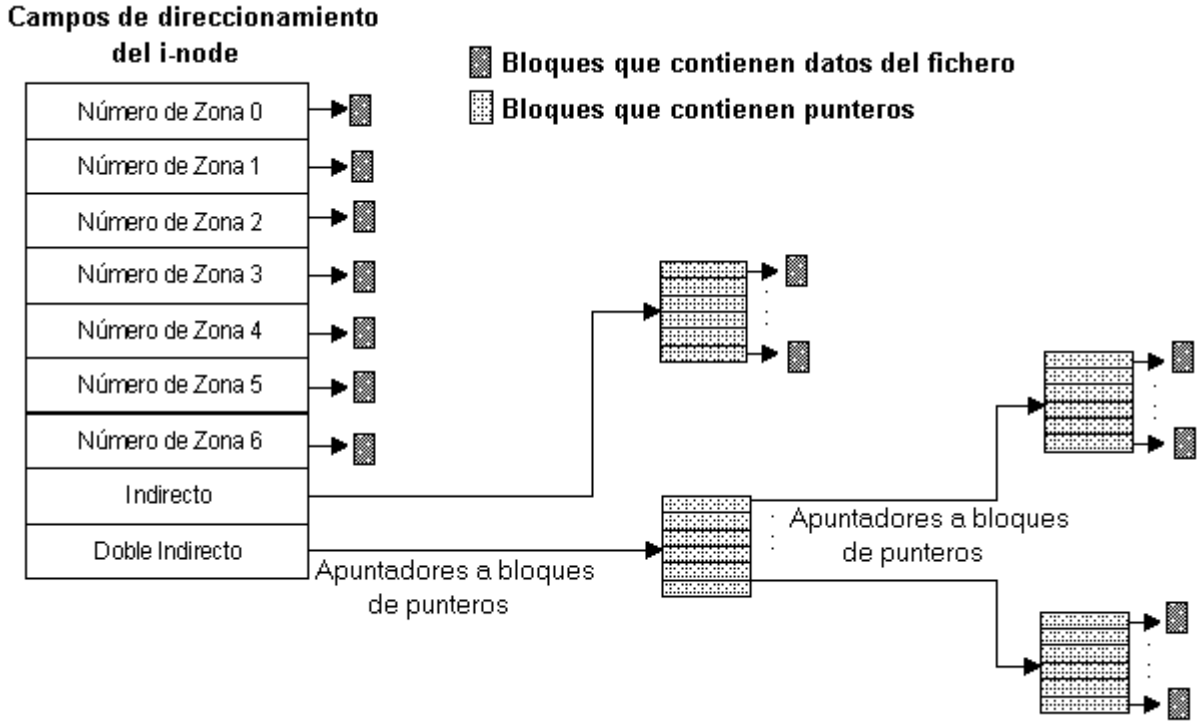
También existe un campo contador por i-node. Si un fichero se abre más de una vez, sólo habrá una copia del i-node en memoria pero su contador será incrementado (*función dup_inode*). El contador se decrementará cuando el fichero se cierre, y cuando valga cero se eliminará de la tabla de i-nodes (si ha sido modificado, se actualizará en el disco).

La función principal del i-node, es **indicar donde se encuentran los bloques de datos**. Los bloques de un fichero de hasta 7Kb se apuntan directamente desde el i-node. Para tamaños mayores se necesitarán indexar por bloques indirectos y doblemente indirectos.

El i-node mantiene también información sobre el tipo de fichero (normal, directorio, especial de bloque, especial de carácter, o conducto de comunicaciones) y las protecciones. El campo *link* indica cuantas entradas de directorio apuntan al fichero. Este campo no debería confundirse con el campo *counter* -sólo presente en memoria- que indica cuantas veces ha sido abierto el fichero (generalmente por procesos diferentes).



Esquema de direccionamiento:



- **i_mount**: Indica si el i-node está montado o no.
- **i_seek**: Se usa para inhibir la lectura anticipada. Cuando advierte que un archivo se está leyendo en forma secuencial, éste intenta leer bloques en la caché antes de que se soliciten, mientras que, para archivos con acceso aleatorio no hay lectura anticipada.
- **i_update:** como obtener la hora del sistema es costoso, cuando se necesite cambiar los campos de tiempo **atime, mtime, ctime,** se marcan los bits del campo **i_update** que los representa mientras está el i-node en memoria. Si este campo es distinto de cero cuando el i-node ha de escribirse en disco, se llama a *update_times*, que actualiza los tres campos mencionados.
- **i_ndzones:** número de zonas directas .
- **i_nindirs:** número de zonas indirectas por bloque indirecto.

ESTRUCTURAS DE DATOS (inode.h)

La estructura de datos que representa a cada inode se encuentra definida en el fichero *"src/fs/inode.h"*. Esta estructura es una tabla que mantiene los i-nodes que están actualmente en uso y que pudieron haber sido abiertos por una llamada al sistema como **open()** o **creat()**; en otros casos puede ser un i-node que el sistema ha tomado por alguna necesidad (como buscar un directorio para una ruta dada).

La primera parte de la estructura mantiene los campos que están presentes en el disco y la segunda parte contiene los campos no presentes en disco pero sí en memoria.
en el fichero *type.h*, donde para el sistema de ficheros V1, la estructura se llama *d1_inode* y para la versión V2 se llama *d2_inode*.

```
EXTERN   struct   inode
{
        /* Campos en disco */
        mode_t   i_mode;          Tipo de fichero, bits de protección, etc.
        uid_t   i_uid;            Identificador de usuario del propietario del fichero.
        off_t   i_size;           Tamaño actual del fichero en bytes.
        time_t   i_mtime;         Cuándo se hizo el último cambio.
        time_t  i_atime;          Hora del útlimo acceso (sólo en V2).
        time_t  i_ctime;          Hora a la que se modificó el estado del i-node (sólo en V2).
        gid_t   i_gid;            Número de grupo.
        nlink_t   i_nlinks;       Número de links (enlaces) que hay al fichero.
        zone_t   i_zone[V2_NR_TZONES];    Números de zonas para accesos directo, indirecto y doble indirecto.

        /* Campos no presentes en disco (si en memoria) */
        dev_t   i_dev;            Dispositivo sobre el que está el i-node.
        ino_t   i_num;            Número del i-node sobre su dispositivo menor.
        int   i_count;            Nº de veces que se abre el archivo; (0 entrada vacía).
        int  i_ndzones;           Zonas directas.
        int  i_nindirs;           Zonas indirectas por bloque indirecto.
        struct super_block *i_sp; Puntero al superbloque del dispositivo del i-node.
        char   i_dirt;            CLEAN o DIRTY (modificado).
        char   i_pipe;            I_PIPE si se trata de un conducto.
        char   i_mount;           Este bit se activa si el fichero está montado.
        char   i_seek;            Activo LSEEK (acceso aleatorio) e inactivo en READ/WRITE (acceso secuencial).
        char   i_update;          Almacena los bits de ATIME, CTIME y MTIME.
} inode[NR_INODES];

#define NIL_INODE  (struct inode *)   Indica la ausencia de la entrada del i-node.
#define NO_PIPE  0                    El i-node no es conducto de comunicación.
#define I_PIPE  1                     El i-node es conducto de comunicación.
#define NO_MOUNT  0                   Fichero no montado.
#define I_MOUNT  1                    Fichero montado.
#define NO_SEEK  0                    La última operación no fue de búsqueda.
#define ISEEK  1                      La última operación fue de búsqueda.
```

INODE.C

Este fichero contiene los procedimientos que manejan la tabla de i-nodes. Vamos a disponer de procedimientos para asignar y desasignar i-nodes, adquirirlos, borrarlos, liberarlos, leerlos y escribirlos en disco. Estos procedimientos son utilizados frecuentemente por el resto del sistema de fichero (FS).

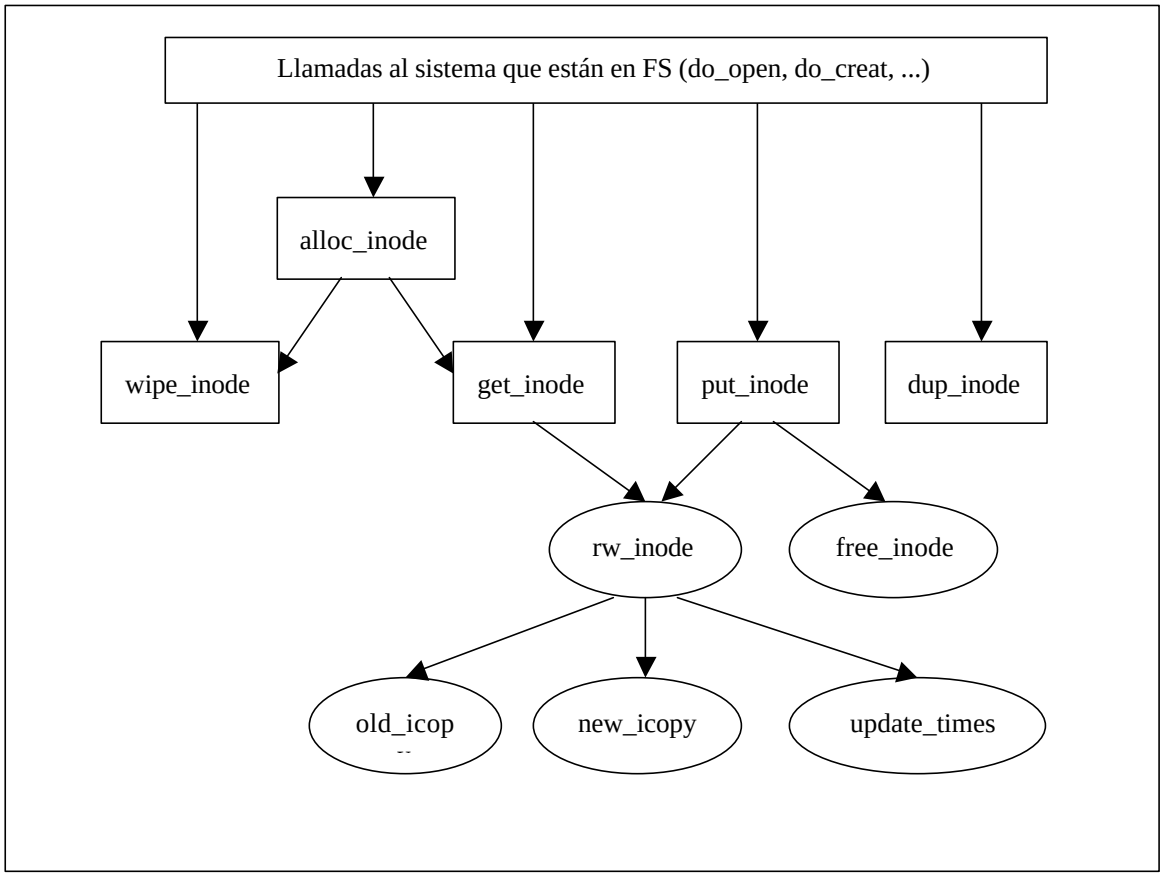
Los puntos de entrada al fichero inode.c son :

- *get_inode* : busca en la tabla de i-nodes uno dado.
- *put_inode* : devuelve el i-node una vez que el procedimiento que lo necesitaba haya acabado con él.
- *alloc_inode* : asigna un nuevo i-node libre a un fichero.
- *wipe_inode* : borrar algunos campos de un i-node.
- *dup_inode* : indica que alguien más está utilizando una entrada en la tabla.

El resto de funciones dan soporte a las anteriores:

- *free_inode* : marca un i-node como disponible para un nuevo fichero.

- *update_times* : actualiza *atime, ctime* y *mtime* que indican el tiempo del último acceso, el tiempo del último cambio de estado del i-node y cuando se hizo el último cambio repectivamente.
- *rw_inode* : se copia una entrada de la tabla del i-node en o desde el disco.
- *old_icopy:* copiar de/a la tabla en memoria a disco (sistema de ficheros V1)
- *new_icopy:* copmiar de/a la tabla en memoria a disco (sistema de ficheros V2)



CÓDIGO

Estos son los ficheros cabecera que usa y las declaraciones prototipos :

```
#include "fs.h"
#include <minix/boot.h>
#include "buf.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "super.h"

FORWARD _PROTOTYPE( void old_icopy, (struct inode *rip, d1_inode *dip,          int direction, int norm));
FORWARD _PROTOTYPE( void new_icopy, (struct inode *rip, d2_inode *dip, int direction, int norm));
```

get_inode

Función

Cuando alguna parte del FS necesita un i-node, éste llama a *get_inode* para adquirirlo.

Parámetros de entrada

dev: Dispositivo donde reside el i-node.

numb: Número del i-node.

Variables más importantes

inode: Vector de i-nodes.

i_count: Contador de uso del i-node.

Procedimientos que utiliza

rw_inode: Se utiliza para leer del disco el i-node.

Algoritmo

- Busca en la tabla de i-nodes (*inode*) el i-node que se necesita y al mismo tiempo busca un hueco libre en la tabla. Si está presente entonces se incrementa el contador de uso (*i_count*) y devuelve un puntero al i-node.
- Si no está presente y no encontró un hueco libre devuelve error indicando que la tabla de i-nodes está llena.
- Si hay un hueco libre, inicializa sus campos adicionales y llama a *rw_inode* para cargar el i-node desde disco a la tabla *inode* en memoria, y devuelve un puntero al i-node.

```
/*===========================================================================*
 *                                get_inode                                  *
 *===========================================================================*/
PUBLIC struct inode *get_inode(dev, numb)
dev_t dev;              /* dispositivo en el que reside el inode */
int numb;                       /* número de inode */
{
        /* Encontrar un hueco en la tabla de inodes, cargar el inode especificado en él
         * devolver un puntero al hueco. Si 'dev' == NO_DEV, devolver un hueco libre*/
        register struct inode *rip, *xp;

        /* buscar la tabla de inodes para (dev,numb) y liberar hueco.*/

        xp = NIL_INODE;

        for (rip = &inode[0]; rip < &inode[NR_INODES]; rip++)
        {
                if (rip->i_count > 0)
                {
                        /* compobar solo los huecos usados para (dev, numb) */
                        if (rip->i_dev == dev && rip->i_num == numb)
                        {
                                rip->i_count++;     /*Este es el inode que estábamos buscado.*/
                                return(rip);        /* (dev, numb) encontrado */
                        }
                } else {
                        xp = rip;  /* recordar este hueco libre para después */
                }
        }
        /* El inode que queremos no está actualmente en uso. */
        /*¿ Hemos encontrado hueco libre ?*/
        if (xp == NIL_INODE)
        {
                /* tabla de inode completamente llena */
                err_code = ENFILE;
                return(NIL_INODE);
        }
        /* Se encontró un hueco de inode. Cargar el inode en él.*/
        xp->i_dev = dev;
        xp->i_num = numb;
        xp->i_count = 1;
        if (dev != NO_DEV) rw_inode(xp, READING);   /* obtener inode de disco.*/
        xp->i_update = 0;                           /* todo está actualizado */

        return(xp);
}
```

put_inode

Función

Devuelve el i-node al disco una vez que el procedimiento que lo necesitaba haya acabado con él.

Parámetros de entrada

rip: Puntero al i-node.

Procedimientos que utiliza

rw_inode : Se utiliza para escribir el i-node en disco.

free_inode : Para eliminar el i-node si borramos el archivo.

truncate (src/fs/link.c): Es una rutina externa. Se usa para liberar todas las zonas del fichero y marcar el i-node como modificado *(dirty).*

Algoritmo

- Comprueba que es un i-node válido.
- Decrementar el contador de uso *i_count*.
- Si *i_count = 0* significa que nadie lo está usando ahora y,
- Si *i_link* = 0 (no hay entradas en ningún directorio que apunten a este fichero) devolvemos todos sus bloques de datos a disco (*truncate*), marcamos el i-node como no asignado y como modificado (*dirty),* y finalmente liberamos el i-node (*free_inode*).
- Si hay entradas en algún directorio que apunten a este fichero (*i_link<>0)* y se trata de un fichero de comunicación *(pipe),* simplemente devolvemos sus bloques de disco *(truncate).*
- Marcamos el i-node como *NO*_PIPE y si fue modificado (*i_dirty = DIRTY*) lo copiamos en disco *(rw_inode).*

```
/*===========================================================================*
 *                                put_inode                                  *
 *===========================================================================*/
PUBLIC void put_inode(rip)
register struct inode *rip;      /* puntero al inode a liberar. */
{
        /* El llamador no va a usar más este inode. Si nadie más lo usa escribirlo inmediatamante
        en disco. Si no tiene links, truncarlo y devolverlo al pool de inodes libres. */

        if (rip == NIL_INODE) return;           /* aquí es más fácil comprobarlo que en el llamador*/
        if (--rip->i_count == 0)                /* i_count == 0 significa que nadie lo usa ahora */
        {
                if ((rip->i_nlinks & BYTE) == 0) {              /* i_nlinks == 0 significa liberar el inode.*/
                        truncate(rip);                          /* devolver todos los bloque de disco */
                        rip->i_mode = I_NOT_ALLOC;              /* borrar el campo I_TYPE  */
                        rip->i_dirt = DIRTY;
                        free_inode(rip->i_dev, rip->i_num);
                } else {
                        if (rip->i_pipe == I_PIPE) truncate(rip);
                }
                rip->i_pipe = NO_PIPE;      /* debería borrarse siempre */
                if (rip->i_dirt == DIRTY) rw_inode(rip, WRITING);
```

```
        }
}
```

alloc_inode

Función

Asigna un i-node libre a un fichero recién creado en un dispositivo, y devuelve un puntero a él.

Parámetros de entrada

dev: Dispositivo que contiene el i-node.

bits: Modo del i-node.

Variables que utiliza

sp: Puntero al superbloque del dispositivo *dev*.

Procedimientos que utiliza

alloc_bit (src/fs/super.c): Asigna un i-node libre (bit a cero) del mapa de bits de i-nodes y devuelve su número.

get_super (src/fs/super.c): Busca el superbloque de un dispositivo dado y devuelve el puntero a él.

get_inode: Para la obtención de un i-node.

wipe_inode: Inicializa varios campos del i-node.

free_bit (src/fs/super.c): Libera un i-node activando su bit del mapa de bits de i-nodes.

Algoritmo

- Se comprueba si el dispositivo es de solo lectura, en cuyo caso devuelve un *NIL_INODE*.
- Tomar un número de i-node libre del mapa de bits de i-nodes. Si no lo encuentra, indica que no quedan i-nodes libres en disco.
- Llamar a *get_inode* para obtener el i-node en la tabla de i-nodes de memoria *(inode)*. Si no hubiera espacio libre en la tabla se libera dicho i-node *(free_bit)*.
- Inicializa varios campos adicionales del i-node ya en la tabla, y llama a *wipe_inode* para que inicialice el resto de los campos.

```
/*===========================================================================*
 *                                alloc_inode                                *
 *===========================================================================*/
PUBLIC struct inode *alloc_inode(dev, bits)
dev_t dev;                      /* dispositivo sobr el que tomar el inode */
mode_t bits;                    /* modo del inode */
{
        /* Tomar un inodo libre en 'dev', y devolver unnpuntero a él.*/
        register struct inode *rip;
        register struct super_block *sp;
        int major, minor, inumb;
        bit_t b;
```

```
	sp = get_super(dev);		/* obtener puntero a superbloque */
	if (sp->s_rd_only)
	{
		/* no puede tomar un inode en un dispositivo de solo lectura */
		err_code = EROFS;
		return(NIL_INODE);
	}

	/* Adquirir un inode del bit map .*/
	b = alloc_bit(sp, IMAP, sp->s_isearch);
	if (b == NO_BIT)
	{
		err_code = ENFILE;
		major = (int) (sp->s_dev >> MAJOR) & BYTE;
		minor = (int) (sp->s_dev >> MINOR) & BYTE;
		printf("Out of i-nodes on %sdevice %d/%d\n",sp->s_dev == ROOT_DEV ? "root " : "", major, minor);
		return(NIL_INODE);
	}

	sp->s_isearch = b;			/* la próxima vez empieza aquí */
	inumb = (int) b;

	/* Intenta adquirir un hueco en la tabla de inodes.*/
	if ((rip = get_inode(NO_DEV, inumb)) == NIL_INODE)
	{
		/* No hay huecos de la tabla de inodes en memoria. Liberar el inodo recién tomado.*/
		free_bit(sp, IMAP, b);
	} else {
		/* Hay un hueco libre. Poner el nodo recién tomado en él.*/
		rip->i_mode = bits;			/*inicializar bits RWX*/
		rip->i_nlinks = (nlink_t) 0;	/* inicalmente no hay links */
		rip->i_uid = fp->fp_effuid;	/* el uid del fichero es el del usuario*/
		rip->i_gid = fp->fp_effgid;	/* ditto group id */
		rip->i_dev = dev;		/* marcar sobre qué dispositivo se encuentra */
		rip->i_ndzones = sp->s_ndzones;	/* número de zonas directas */
		rip->i_nindirs = sp->s_nindirs;	/* número de zonas indirectas por bloque*/
		rip->i_sp = sp;			/* puntero a superbloque */

		/* Los campos no borrados se borran en  wipe_inode(). Se han puesto ahí porque truncate() necesita
		borrar los mismos campos si el fichero resulta estar abierto mientras se trunca. Se ahorra espacio no
		repitiendo el código dos veces.*/

		wipe_inode(rip);
	}

	return(rip);
}
```

wipe_inode

Función

Borra algunos campos del i-node. La invoca *alloc_inode* cuando se va a asignar un nuevo i-node, y *truncate* cuando se va a truncar un nodo ya existente.

Parámetros de entrada

rip : Puntero al i-node.

```
/*===========================================================================*
 *                                wipe_inode                                 *
 *===========================================================================*/
PUBLIC void wipe_inode(rip)
register struct inode *rip;      /* el inode que se va a borrar */
{

        /* Borrar algunos campos en el inode. Esta función se llama desde alloc_inode()
        cuando se va a tomar un nuevo inode, y desde truncate(), cuando se va a truncar un inode existente.*/

        register int i;
        rip->i_size = 0;
        rip->i_update = ATIME | CTIME | MTIME;          /* actualizar todos los times después*/
        rip->i_dirt = DIRTY;
        for (i = 0; i < V2_NR_TZONES; i++)
                rip->i_zone[i] = NO_ZONE;
}
```

free_inode

Función

Retorna un i-node al conjunto de nodos que no están asignados, o sea, marca un i-node como disponible para un nuevo fichero en el mapa de bits de i-nodes. A su vez, el registro del superbloque que indica el primer i-node no usado, se actualiza.

Parámetros de entrada

dev: Dispositivo en el que está el i-node.

inumb: Número del i-node que se liberará.

Variables que utiliza

sp: Puntero al superbloque del dispositivo *dev*.

Procedimientos que utiliza

get_super (src/fs/super.c): Busca el superbloque de un *dev* dado y devuelve el puntero a él.

free_bit (src/fs/super.c): Libera un i-node activando su bit del mapa de bits.

```
/*===========================================================================*
 *                                free_inode                                 *
 *===========================================================================*/
PUBLIC void free_inode(dev, inumb)
dev_t dev;                      /* en qué dispositivo está el inode */
ino_t inumb;                    /* número del inode a ser liberado */
{
        /* Devolver un inode al pool de inodes libres */
        register struct super_block *sp;  bit_t b;

        /* Localizar el superbloque apropiado .*/
        sp = get_super(dev);
        if (inumb <= 0 || inumb > sp->s_ninodes) return;
        b = inumb;
        free_bit(sp, IMAP, b);
        if (b < sp->s_isearch) sp->s_isearch = b;
}
```

update_times

Función

El estándar necesita varias llamadas al sistema para actualizar *atime, ctime* o *mtime*. Como para actualizar el tiempo se necesita enviar un mensaje a la tarea del reloj (un asunto caro) los tiempos se marcan para una actualización activando los bits en *i_update*. Cuando se hace un *stat, fstat* o *sync* o se libera un i-node se debe llamar a *update_times* para completar los tiempos.

Parámetros de entrada

rip: Puntero al i-node.

Procedimientos que utiliza

clock_time: Rutina que obtiene el tiempo en segundos desde 1/1/1970. Se encuentra en *src/fs/utility.c*.

```
/*===========================================================================*
 *                                update_times                               *
 *===========================================================================*/

PUBLIC void update_times(rip)
register struct inode *rip;     /* puntero al inode a ser leido o escrito*/
{
        /* Se requieren varias llamadas al sistema para actualizar atime, ctime, y mtime. Dado que actualizar un time
        requiere enviar un mensaje a la tarea del reloj –cosa costosa- los times son marcados para actualizarlos
        poniendo bits en i_update. Cuando se hace un stat, fstat, o sync o cuando se libera un i-node, se puede llamar a
        update_times() para rellenar realmente los times.*/

        time_t cur_time;
        struct super_block *sp;

        sp = rip->i_sp;                 /* obtener puntero al superbloque.*/
        if (sp->s_rd_only) return;      /*no hay actualizaciones para los ficheros de sólo lectura */
```

```
        cur_time = clock_time();                /*obtener la hora del sistema*/
        if (rip->i_update & ATIME) rip->i_atime = cur_time;
        if (rip->i_update & CTIME) rip->i_ctime = cur_time;
        if (rip->i_update & MTIME) rip->i_mtime = cur_time;
        rip->i_update = 0;                              /* ahora están todos actualizados */
}
```

rw_inode

Función

Copia un i-node de la tabla de i_nodes en memoria en el disco, o bien copia desde el disco un i-node en la tabla en memoria.

Parámetros de entrada

rip : Puntero al i-node.

rw_flag : Indica si se trata de una lectura o una escritura.

Variables que utiliza

sp : Puntero al superbloque del dispositivo *dev*.

bp : Puntero a los bloques almacenados en el vector *buf*.

Procedimientos que utiliza

get_super: Busca el dispositivo en la tabla del superbloque y devuelve el puntero a él. Se encuentra en super.c

get_block: Coge un bloque para leer o escribir de la reserva. Está en cache.c.

old_icopy: Convierte la información para el sistema de ficheros V1.

new_icopy: Convierte la información para el sistema de ficheros V2.

update_times: Función explicada previamente.

put_block: Devuelve un bloque a la lista de bloques disponibles. Está en cache.c.

Algoritmo

- Calcular que bloque tiene el i-node requerido *(get_super).*
- Leer el bloque llamando a *get_block.*
- Si se trata de una escritura se actualizan los campos de tiempos llamando a *update_times.*
- Según la versión del sistema de ficheros (V1 o V2) se usará *old_icopy* o bien *new_icopy* para leer o escribir en disco.

```
/*===========================================================================*
 *                                rw_inode                                   *
 *===========================================================================*/
PUBLIC void rw_inode(rip, rw_flag)
register struct inode *rip;     /* puntero al inode a ser leido/escrito.*/
int rw_flag;                            /* READING o WRITING */
{
        /* Una entrada de la tabla de inodes  será copiada a o desde disco.*/
        register struct buf *bp;
```

```
        register struct super_block *sp;
        d1_inode *dip;
        d2_inode *dip2;
        block_t b, offset;

        /* Obtener el bloque donde reside el inode */
        sp = get_super(rip->i_dev);  /* obtener puntero al superbloque */
        rip->i_sp = sp;                         /* el inode debe contener el puntero al superbloque*/
        offset = sp->s_imap_blocks + sp->s_zmap_blocks + 2;
        b = (block_t) (rip->i_num - 1)/sp->s_inodes_per_block + offset;
        bp = get_block(rip->i_dev, b, NORMAL);
        dip  = bp->b_v1_ino + (rip->i_num - 1) % V1_INODES_PER_BLOCK;
        dip2 = bp->b_v2_ino + (rip->i_num - 1) % V2_INODES_PER_BLOCK;

        /* Realizar la lectura o la escritura*/
        if (rw_flag == WRITING)
        {
                if (rip->i_update) update_times(rip);    /*los tiempos han de actualizarse*/
                if (sp->s_rd_only == FALSE) bp->b_dirt = DIRTY;
        }
        /* Copiar el inode del bloque de disco a la tabla residente o viceversa.
        Si el cuarto parámetro de los de abajo es falso, se intercambian los bytes.*/
        if (sp->s_version == V1)
                old_icopy(rip, dip,  rw_flag, sp->s_native);
        else
                new_icopy(rip, dip2, rw_flag, sp->s_native);

        put_block(bp, INODE_BLOCK);
        rip->i_dirt = CLEAN;
}
```

dup_inode

Función

Esta rutina es una forma simplificada de la rutina *get_inode* para el caso en el que el puntero al i-node sea conocido. Tan solo incrementa el número de referencias.

Parámetro de entrada

ip: Puntero al i-node.

```
/*===========================================================================*
 *                                dup_inode                                  *
 *===========================================================================*/
PUBLIC void dup_inode(ip)
struct inode *ip;               /* Inode a duplicar.*/
{
        /* Esta rutina es una forma simplificada de get_inode() para el caso en el que ya se conoce el puntero al inode. */

        ip->i_count++;
}
```

old_icopy

Función

EL disco V1.x de IBM , el disco V1.x 68000, y el disco V2 (iguales en IBM que 68000) tinene todos diseños diferentes de inodes. Cuando se lee o escribe un inode, esta rutina maneja la conversión de manera que la información de la table de inodes sea independiente de la estructura del discode donde viniera el inode. La rutina old_copy copia a y desde discos V1.

Parámetros de entrada

rip : Puntero al i-node.

dip : Puntero a la estructura d1_inode.

direction: Indica lectura o escritura en el disco.

norm: Si es TRUE, no intercambiar bytes.

Procedimientos que utiliza

conv2: Cambia el orden de los bytes de una palabra de 16 bits (del orden usado del 8086 al 68000).

conv4: Cambia el orden de los bytes de una palabra de 32 bits (del orden usado del 8086 al 68000).

```
/*===========================================================================*
 *                                old_icopy                                  *
 *===========================================================================*/
PRIVATE void old_icopy(rip, dip, direction, norm)
register struct inode *rip;     /* puntero a la estructura de inode residente*/
register d1_inode *dip;         /*puntero al la estructura de inode d1_inode */
int direction;                          /* READING (a disco) o WRITING (desde disco) */
int norm;                                       /* TRUE = no intercambiar bytes; FALSE = intercambiarlos*/

{
        int i;

        if (direction == READING) {
                /* Copiar inode V1.x a la tabla de memoria, intercambiando los bytes si fuera necesario. */

                rip->i_mode    = conv2(norm, (int) dip->d1_mode);
                rip->i_uid     = conv2(norm, (int) dip->d1_uid );
                rip->i_size    = conv4(norm,       dip->d1_size);
                rip->i_mtime   = conv4(norm,       dip->d1_mtime);
                rip->i_atime   = rip->i_mtime;
                rip->i_ctime   = rip->i_mtime;
                rip->i_nlinks  = (nlink_t) dip->d1_nlinks;          /* 1 char */
                rip->i_gid     = (gid_t) dip->d1_gid;               /* 1 char */
                rip->i_ndzones = V1_NR_DZONES;
                rip->i_nindirs = V1_INDIRECTS;

                for (i = 0; i < V1_NR_TZONES; i++)
```

```
                        rip->i_zone[i] = conv2(norm, (int) dip->d1_zone[i]);
        } else {
                /* Copiar inode V1.x a disco desde la tabla en memoria.*/

                dip->d1_mode   = conv2(norm, (int) rip->i_mode);
                dip->d1_uid    = conv2(norm, (int) rip->i_uid );
                dip->d1_size   = conv4(norm,       rip->i_size);
                dip->d1_mtime  = conv4(norm,       rip->i_mtime);
                dip->d1_nlinks = (nlink_t) rip->i_nlinks;           /* 1 char */
                dip->d1_gid    = (gid_t) rip->i_gid;                /* 1 char */
                for (i = 0; i < V1_NR_TZONES; i++)
                        dip->d1_zone[i] = conv2(norm, (int) rip->i_zone[i]);
        }
}
```

new_icopy

Función

La misma que *old_icopy* pero para el diseño de disco V2.

Parámetros de entrada

rip : Puntero al i-node.

dip : Puntero a la estructura d2_inode.

direction: Indica lectura o escritura en el disco.

norm: Si es TRUE, no intercambiar bytes.

Procedimientos que utiliza

conv2: Cambia el orden de los bytes de una palabra de 16 bits (del orden usado del 8086 al 68000).

conv4: Cambia el orden de los bytes de una palabra de 32 bits (del orden usado del 8086 al 68000).

```
/*===========================================================================*
 *                                new_icopy                                  *
 *===========================================================================*/
PRIVATE void new_icopy(rip, dip, direction, norm)
register struct inode *rip;             /* puntero a la estructura inode en memoria.*/
register d2_inode *dip;                 /* puntero a la estructura d2_inode. */
int direction;                                  /* READING (de disco) o WRITING (a disco) */
int norm;                                               /* TRUE = no intercambiar bytes; FALSE = intercambiar*/

{
        int i;

        if (direction == READING) {
                /* Copiar inode V2.x a la tabla en memria, intercambiando los bytes si fuera necesario.*/

                rip->i_mode    = conv2(norm,dip->d2_mode);
                rip->i_uid     = conv2(norm,dip->d2_uid );
                rip->i_nlinks  = conv2(norm,(int) dip->d2_nlinks);
```

```
                rip->i_gid     = conv2(norm,(int) dip->d2_gid );
                rip->i_size    = conv4(norm,dip->d2_size);
                rip->i_atime   = conv4(norm,dip->d2_atime);
                rip->i_ctime   = conv4(norm,dip->d2_ctime);
                rip->i_mtime   = conv4(norm,dip->d2_mtime);
                rip->i_ndzones = V2_NR_DZONES;
                rip->i_nindirs = V2_INDIRECTS;
                for (i = 0; i < V2_NR_TZONES; i++)
                        rip->i_zone[i] = conv4(norm, (long) dip->d2_zone[i]);
        } else {

                /* Copiar  inode V2 al disco desde la tabla en memoria .*/

                dip->d2_mode   = conv2(norm,rip->i_mode);
                dip->d2_uid    = conv2(norm,rip->i_uid );
                dip->d2_nlinks = conv2(norm,rip->i_nlinks);
                dip->d2_gid    = conv2(norm,rip->i_gid );
                dip->d2_size   = conv4(norm,rip->i_size);
                dip->d2_atime  = conv4(norm,rip->i_atime);
                dip->d2_ctime  = conv4(norm,rip->i_ctime);
                dip->d2_mtime  = conv4(norm,rip->i_mtime);
                for (i = 0; i < V2_NR_TZONES; i++)
                        dip->d2_zone[i] = conv4(norm, (long) rip->i_zone[i]);
        }
}
```

CUESTIONES.

Diferencia entre el campo link y el campo contador.

1) Link es un campo del nodo-i que está presente tanto en el disco como en memoria, mientras que el campo contador se añade al nodo-i sólo cuando el nodo-i es traído a memoria.

2) El valor del campo link indica cuantos directorios hacen referencia al nodo-i. Si su valor es 0 significa que el fichero no es referenciado por nadie y por lo tanto se puede borrar.

3) El valor del campo contador indica el número de referencias sobre el nodo-i del fichero abierto. Si su valor es 0 significa que el fichero se puede cerrar, y por tanto su nodo-i puede ser llevado a disco.

¿Nº de bloques de datos referenciables para un bloque de 1k y punteros de 16 bits.?

Directos : 7 bloques

Indirectos : 512 bloques

Dobles Indirectos : 512 punteros x 512 bloques = 262144

Total = 7 + 512 + 262144 = 262. 663.

¿ Cómo se intenta disminuir el costo de enviar mensajes a la tarea del reloj ?

Cuando se realiza una operación que implica modificación se indica en el campo i_update y posteriormente cuando se realiza un stat, fstat o sync o se libera un nodo se llama a update_times para completar los tiempos.