Mount / Umount

Carmelo Alexis Acosta Ojeda
Javier Verdú Mulá

ÍNDICE

Introducción

Un concepto clave del diseño de software de E/S es la independencia del dispositivo, es decir, emplear las mismas instrucciones para leer un fichero sin importar si éste reside en el disco duro o en el CDRom. De esta manera queda la responsabilidad del tratamiento específico de cada dispositivo al sistema operativo, al igual que los posibles errores derivados de este subnivel.

Para entender mejor este objetivo, observamos las diferencias entre un sistema operativo que implemente el montaje de sistema de archivos (como MINIX) y uno que no lo haga (MSDOS).

En primer lugar, un dispositivo puede estar montado sobre un directorio determinado. Por tanto no hay distinción entre copiar un fichero a un directorio del disco duro o a un disquete. Sin embargo, de la otra forma, deberíamos especificar la ruta partiendo de la unidad asociada a dicho dispositivo. Así pues, aún siendo una incomodidad para el usuario, mientras estemos trabajando sobre un equipo que tiene un reducido número de dispositivos se puede soportar la especificación de la letra de la unidad asociada a la disquetera que queremos acceder. Ahora bien, si el número aumenta en gran medida resulta muy incomodo. En cambio, si montamos el sistema de archivos en un directorio es muchísimo más cómodo y sencillo de utilizar.

Funcionamiento Interno y Desarrollo

Tanto ***Mount*** como ***Umount*** afectan al Sistema de Archivos como un todo y permiten que sistemas de archivos independientes de dispositivos menores diferentes se «peguen» para formar un solo árbol de nominación.

En la gráfica 1 se muestran los pasos principales del mount. Los nodos de los dos árboles deben ser interpretados como los i-nodes asociados a cada uno de los directorios. Los pasos se pueden describir como sigue:

1) Cargamos el dispositivo en memoria.

   Ello implica cargar el Superbloque en la tabla de Superbloques, cargar los mapas de bits de zonas y de nodos-i libres y cargar en memoria el inode raíz de dicho dispositivo.

2) Indicamos que se va a montar un sistema de ficheros sobre el inode indicado por "rip".

   Así, modificamos el campo "i_mount", poniéndolo al valor "I_MOUNT".

3) Modificamos el Superbloque del dispositivo montado.

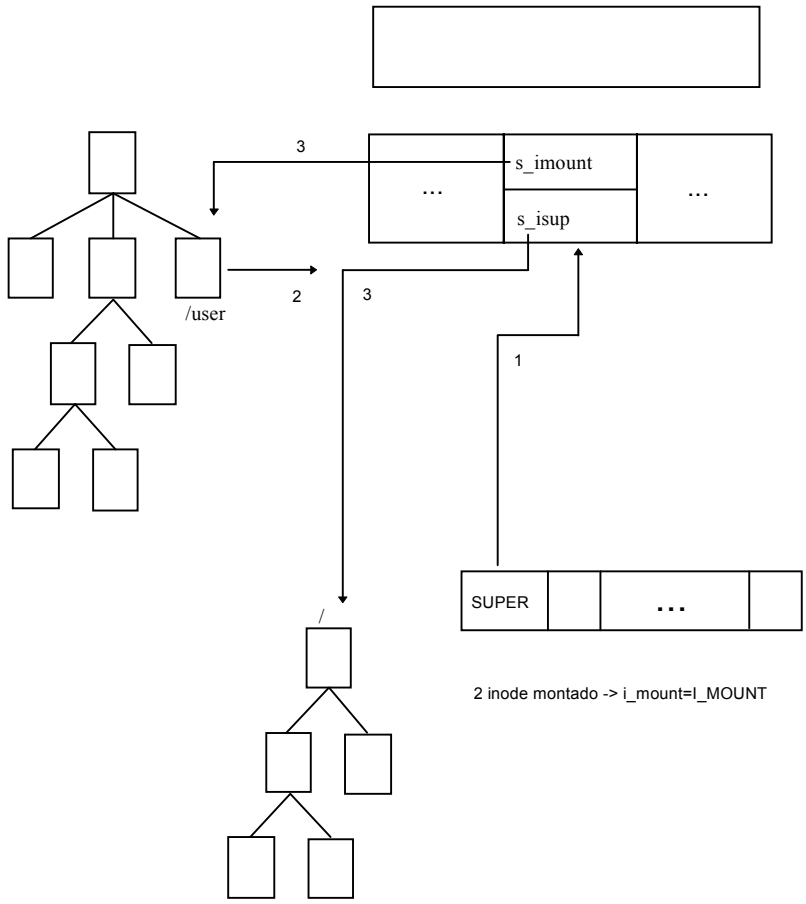
   Hacemos que el campo "s_imount" tome el valor del inode sobre el que vamos a montar el dispositivo (el inode "/user") y el campo "s_isup" tome el valor del inode root del dispositivo a montar.

Para apreciar como influye, en la búsqueda de un archivo determinado, el hecho de que éste cuelgue directamente del nodo raíz ya montado, o pertenezca a un sistema de archivo que acabamos de montar, podemos observar los pasos seguidos en las GRÁFICAS 2 y 3.

GRÁFICA 1

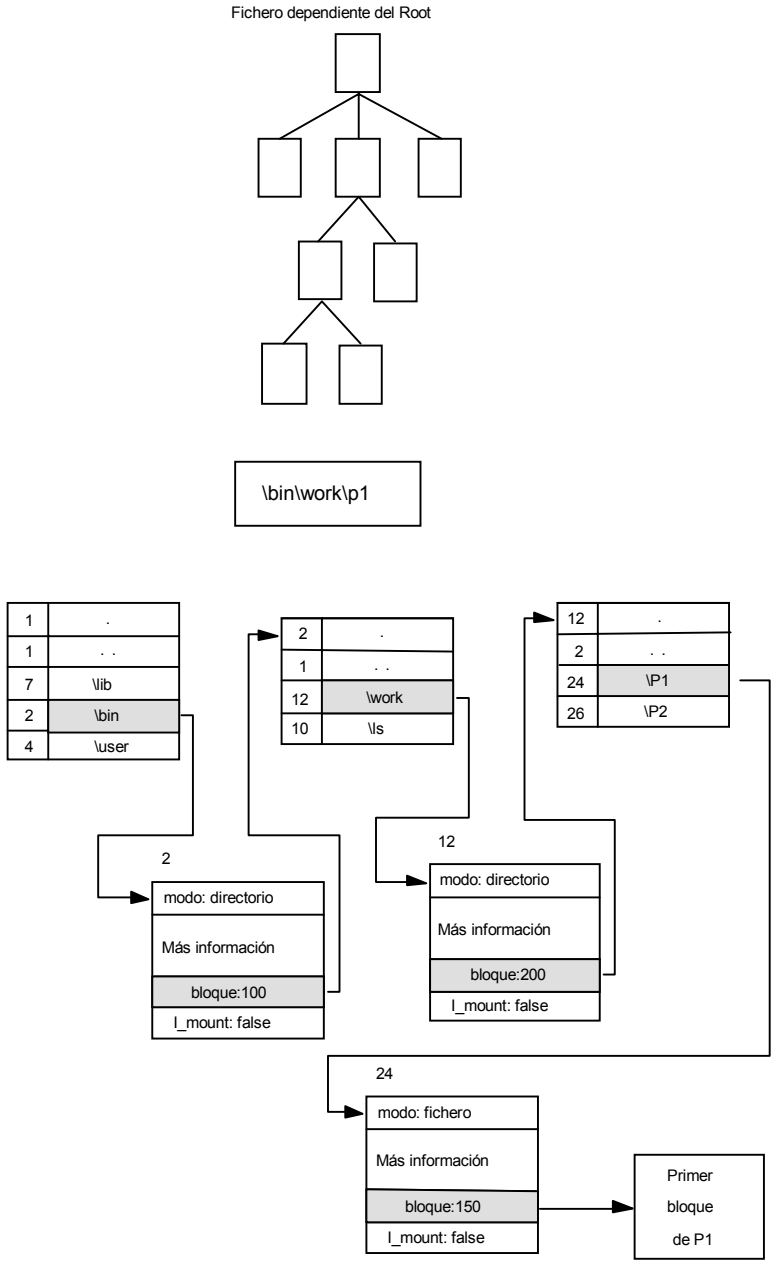
Pasos del montaje de un sistema de archivo

\etc\mount \dev\fd1 \user



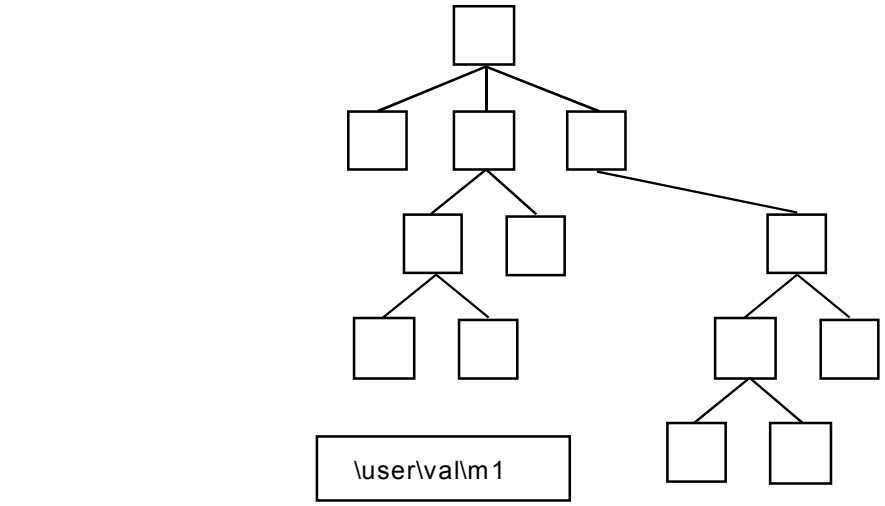
2 inode montado -> i_mount=I_MOUNT

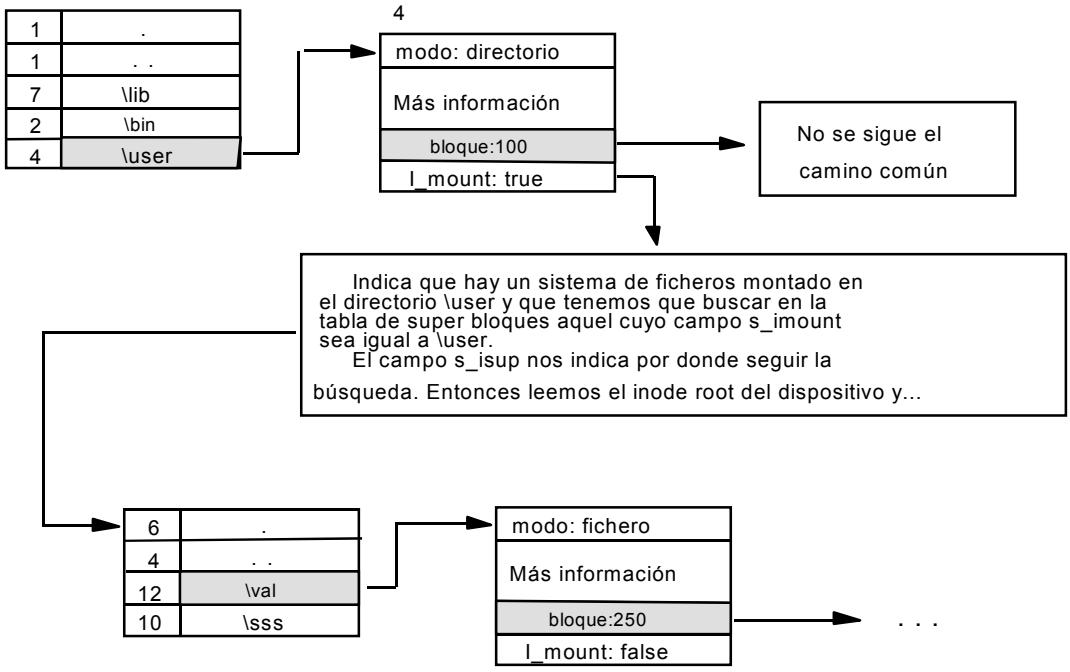
GRÁFICA 2


Fichero dependiente del Root
\bin\work\p1
1 .
1 . .
7 \lib
2 \bin
4 \user
2 .
1 . .
12 \work
10 \ls
12 .
2 . .
24 \P1
26 \P2
2
modo: directorio
Más información
bloque:100
l_mount: false
12
modo: directorio
Más información
bloque:200
l_mount: false
24
modo: fichero
Más información
bloque:150
l_mount: false
Primer
bloque
de P1

GRÁFICA 3

Fichero dependiente de un sistema de archivo montado





do_mount

Función

El archivo MOUNT.C realiza el proceso de montaje por medio de **do_mount.** Los errores que pueden producirse mientras se produce el montaje son los siguientes:

- El archivo especial no es un dispositivo de bloque.
- El archivo especial ya está montado.
- El sistema de archivo que se va a montar tiene un número mágico deteriorado (por ejemplo, es uno de MSDOS).
- El sistema de archivo que se va a montar no es válido, no existe o es un archivo especial.
- No hay espacio para los mapas de bits del sistema de archivo montado.
- No hay espacio para el Superbloque del sistema de archivo montado.
- No hay espacio para el inodo raiz del sistema de archivo montado.

Valores de retorno

EPERM	Es un error invocado cuando el mount no lo ejecuta el superusuario.
EBUSY	Cuando el directorio en donde se montará el nuevo sistema de ficheros está ocupado. O cuando el dispositivo ya está montado.
ENFILE	No hay ninguna entrada en la tabla de Superbloques disponible.
EINVAL	El Superbloque no es válido.
EISDIR	El lugar donde se quiere montar el nuevo F.S. no es un directorio o el inode raíz no es un directorio.
ENOTBLK	Cuando el 'path' para el que se quiere saber su numero de dispositivo no es un bloque especial.
NO_DEV	Fallo al intentar acceder al dispositivo, o dispositivo inexistente.
err_code	Error por corrupción de información u otros inconvenientes.
OK	Correcto.

Estructuras de Datos

Este programa trabaja con la estructura tabla de Superbloques y la inode. No utiliza todos los campos de ellas, sino sólo algunos. De la **Tabla de Superbloque** usará los siguientes:

s_dev:	Dispositivo al cual pertenece el Superbloque.
s_imount:	Inode sobre el que está montado el dispositivo.
s_isup:	Inode del directorio raíz del sistema de archivo contenido en el dispositivo.
s_rd_only:	Si su valor es 1 indica que el sistema de archivo montado es sólo de lectura.

De la estructura **Inode** usarán los siguientes campos:

i_count:	Número de veces que es usado el inode. Si su valor es cero indica que la ranura está libre.
i_mode:	Características del archivo (tipo, protección, etc).
i_mount:	Su valor es I_MOUNT si un sistema de archivo está montado sobre él.

Además utiliza:

user_path:	Almacena parámetros de llamada ocasionalmente

Procedimientos que utiliza

Nombre	Descripción	Fichero
Invalidate	Purga todos los bloques de algún dispositivo de la reserva.	cache.c
fetch_name	Obtiene el nombre de la trayectoria del espacio de usuario.	utility.c
read_super	Transfiere un Superbloque entre la memoria y disco.	super.c
eat_path	Devuelve el inode correspondiente a un path.	path.c
get_inode	Captura un inode en la memoria.	inode.c
put_inode	Devuelve un inode que ya no se necesita.	inode.c
do_sync		misc.c

do_mount

```
/* Este fichero lleva a cabo las llamadas al sistema MOUNT y
 * UMOUNT.
 *
 * Los puntos de entrada al fichero son:
 *
 * do_mount:  monta un sistema de fichero ejecutando la llamada
 *            al sistema MOUNT.
 *
 * do_umount: desmonta un sistema de archivo ejecutando la
 *            llamada al sistema UMOUNT.
 */

/*===========================================================*
 *                do_mount                                   *
 *===========================================================*/
```

Los ficheros que se incluyen son:

```
#include "fs.h"
#include <fcntl.h>
#include <minix/com.h>
#include <sys/stat.h>
#include "buf.h"
#include "dev.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "param.h"
#include "super.h"

PRIVATE message dev_mess;

FORWARD _PROTOTYPE( dev_t name_to_dev, (char *path)                    );

/*===========================================================*
 *                                do_mount                   *
 *===========================================================*/


PUBLIC int do_mount()
{
/* Realiza la llamada del sistema mount(name, mfile, rd_only) */

  register struct inode *rip, *root_ip;
  struct super_block *xp, *sp;
```

```
dev_t dev;
mode_t bits;
int rdir, mdir;          /* VERDADERA si {root|mount} es un directorio */
int r, found, major, task;

/* Solo el Superusuario puede realizar el MOUNT. */
if (!super_user) return(EPERM);

/* Si 'name' no es un bloque especial retorna un error. */
if (fetch_name(name1, name1_length, M1) != OK) return(err_code);
if ( (dev = name_to_dev(user_path)) == NO_DEV) return(err_code);

/* Analiza la tabla de Superbloques para comprobar que el dispositivo a montar no esta
montado y busca una entrada libre en la misma.*/
sp = NIL_SUPER;
found = FALSE;
for (xp = &super_block[0]; xp < &super_block[NR_SUPERS]; xp++) {
        if (xp->s_dev == dev) found = TRUE;  /* Ya esta montado */
        if (xp->s_dev == NO_DEV) sp = xp;    /* Guarda una entrada libre. */
}
if (found) return(EBUSY);     /* Dispositivo ya montado */
if (sp == NIL_SUPER) return(ENFILE);        /* No hay entrada disponible */

dev_mess.m_type = DEV_OPEN;                 /* Operación a realizar */
dev_mess.DEVICE = dev;                /* Indica el dispositivo. */
if (rd_only) dev_mess.COUNT = R_BIT;
else  dev_mess.COUNT = R_BIT|W_BIT;

major = (dev >> MAJOR) & BYTE;
if (major <= 0 || major >= max_major) return(ENODEV);
task = dmap[major].dmap_task;                 /* Número de tarea */
(*dmap[major].dmap_open)(task, &dev_mess);
if (dev_mess.REP_STATUS != OK) return(EINVAL);

/* Rellenamos campos del Superbloque. */
sp->s_dev = dev;              /* read_super() necesita conocer el dispositivo en cuestión. */
r = read_super(sp);



/* Comprobamos que es un Sistema de Ficheros Minix. */
if (r != OK) {
        dev_mess.m_type = DEV_CLOSE;
        dev_mess.DEVICE = dev;
        (*dmap[major].dmap_close)(task, &dev_mess);
        return(r);
}

/* Tomamos el i-node del directorio donde se va a montar el nuevo dispositivo. */
if (fetch_name(name2, name2_length, M1) != OK) {
```

```
        sp->s_dev = NO_DEV;
        dev_mess.m_type = DEV_CLOSE;
        dev_mess.DEVICE = dev;
        (*dmap[major].dmap_close)(task, &dev_mess);
        return(err_code);
}
if ( (rip = eat_path(user_path)) == NIL_INODE) {
        sp->s_dev = NO_DEV;
        dev_mess.m_type = DEV_CLOSE;
        dev_mess.DEVICE = dev;
        (*dmap[major].dmap_close)(task, &dev_mess);
        return(err_code);
}

/* Dicho directorio debe estar vacío. */
r = OK;
if (rip->i_count > 1) r = EBUSY;

/* No debe ser especial. */
bits = rip->i_mode & I_TYPE;
if (bits == I_BLOCK_SPECIAL || bits == I_CHAR_SPECIAL) r = ENOTDIR;

/* Tomamos el i-node raíz del Sistema de Ficheros a montar. */
root_ip = NIL_INODE;                    /* Nos aseguramos que r=OK */
if (r == OK) {
        if ( (root_ip = get_inode(dev, ROOT_INODE)) == NIL_INODE) r = err_code;
}
if (root_ip != NIL_INODE && root_ip->i_mode == 0) r = EINVAL;
/* Comprobamos que las características de 'rip' y 'root_ip' son compatibles. */
if (r == OK) {
        mdir = ((rip->i_mode & I_TYPE) == I_DIRECTORY);  /* VERDAD si directorio */
        rdir = ((root_ip->i_mode & I_TYPE) == I_DIRECTORY);
        if (!mdir && rdir) r = EISDIR;
}
/* Si se produce error, retornamos el Superbloque y ambos i-nodes y liberamos los
mapas de  memoria. */
if (r != OK) {
        put_inode(rip);
        put_inode(root_ip);
        (void) do_sync();
        invalidate(dev);

        sp->s_dev = NO_DEV;
        dev_mess.m_type = DEV_CLOSE;
        dev_mess.DEVICE = dev;
        (*dmap[major].dmap_close)(task, &dev_mess);
        return(r);
}

/* Todas las comprobaciones están hechas. Procedemos a montar el Sistema de
```

```
  Ficheros. */
  rip->i_mount = I_MOUNT;    /* Indica que tiene un F.S. montado sobre él. */
  sp->s_imount = rip;
  sp->s_isup = root_ip;
  sp->s_rd_only = rd_only;
  return(OK);
}
```

do_umount

Función

Este procedimiento realiza el proceso de desmontar un sistema de archivo.

Estructuras de Datos

Al igual que el do_mount, trabaja con la tabla de Superbloques y con la estructura inode. De la **tabla de Superbloques** utiliza los siguientes campos:

s_dev:	Indica a qué dispositivo pertenece el Superbloque.
s_imount:	Inode sobre el que está montado el dispositivo.
s_isup:	Inode del directorio raíz del sistema de archivo montado.

De la estructura **Inode** utiliza los siguientes campos:

i_count:	Número de veces que se usa el inode.
i_dev:	Indica sobre qué dispositivo se encuentra el inode.
i_mount:	Su valor es I_MOUNT si el archivo está montado.

Procedimientos que utiliza

Nombre	Descripción	Fichero
Invalidate	Purga todos los bloques de algún dispositivo de la reserva.	cache.c
fetch_name	Obtiene el nombre de la trayectoria del espacio de usuario.	utility.c
do_sync	Realiza la llamada al sistema SYNC.	misc.c

do_umount

```
/*===========================================================*
*                          do_umount                        *
*===========================================================*/

PUBLIC int do_umount()
{
/* Realiza la llamada del sistema UMOUNT(name) */

  register struct inode *rip;
  struct super_block *sp, *sp1;
  dev_t dev;
  int count;
  int major, task;

  /* Solo el Superusuario puede realizar el UMOUNT. */
  if (!super_user) return(EPERM);

  /* Si 'name' no es un tipo de fichero especial retornamos un error. */
  if (fetch_name(name, name_length, M3) != OK) return(err_code);
  if ( (dev = name_to_dev(user_path)) == NO_DEV) return(err_code);

  /* Comprobamos si el dispositivo montado esta ocupado. */
  count = 0;
  for (rip = &inode[0]; rip< &inode[NR_INODES]; rip++)
        if (rip->i_count > 0 && rip->i_dev == dev) count += rip->i_count;
  if (count > 1) return(EBUSY); /* No se puede desmontar un F.S. ocupado */

  /* Encontramos el Superbloque del dispositivo. */
  sp = NIL_SUPER;
  for (sp1 = &super_block[0]; sp1 < &super_block[NR_SUPERS]; sp1++) {
        if (sp1->s_dev == dev) {
                sp = sp1;
                break;
        }
  }
 (void) do_sync();        /* Desalojamos cualquier bloque de cache fuera de la memoria */
  invalidate(dev);    /*Invalidamos las entradas de la cache asociadas a este dispositivo. */
  if (sp == NIL_SUPER) return(EINVAL);

  major = (dev >> MAJOR) & BYTE;   /* Dispositivo mayor */
  task = dmap[major].dmap_task;          /* Numero de tarea */
  dev_mess.m_type = DEV_CLOSE;              /* Cerramos el dispositivo. */
  dev_mess.DEVICE = dev;
```

```
  (*dmap[major].dmap_close)(task, &dev_mess);

  /* Terminamos de desmontar el dispositivo. */
  sp->s_imount->i_mount = NO_MOUNT;       /* I-node retorna a la normalidad. */
  put_inode(sp->s_imount);      /* Libera los i-nodes montados */
  put_inode(sp->s_isup);        /* Libera el i-node raíz del Sistema de Ficheros montado. */
  sp->s_imount = NIL_INODE;
  sp->s_dev = NO_DEV;
  return(OK);
}
```

name_to_dev

Función

Este procedimiento toma un fichero especial de bloque, determina su nodo_i y extrae sus números de dispositivo mayor y menor. Estos se encuentran almacenados en el nodo_i mismo, en el lugar donde normalmente se dirigiría la primera zona. Esta ranura está disponible porque los archivos especiales no tienen zonas.

Parámetros

path: Puntero al nombre del path del fichero especial de bloque.

Devuelve un número de dispositivo o un código de error.

Procedimientos que utiliza

Nombre	Descripción	Fichero
eat_path	Devuelve el nodo_i correspondiente a un path.	path.c
put_inode	Devuelve un inode que ya no se necesita.	inode.c

name_to_dev

```
PRIVATE dev_t name_to_dev(path)
char *path;                         /* Puntero al path name */
{
/*Convierte el fichero especial de bloque contenido en 'path' a un número de dispositivo.
Si el 'path' no es un fichero de bloque especial devuelve un código de error.*/

  register struct inode *rip;
  register dev_t dev;

  /* Si 'path' no puede ser abierto se devuelve un error. */
  if ( (rip = eat_path(path)) == NIL_INODE) return(NO_DEV);

  /* Si 'path' no es un fichero especial de bloque, liberamos el i-node que se obtuvo con
  'eat_path' y devolvemos un error. */
  if ( (rip->i_mode & I_TYPE) != I_BLOCK_SPECIAL) {
          err_code = ENOTBLK;
          put_inode(rip);
          return(NO_DEV);
  }

  /* Extraer el número de dispositivo y liberar el i-node. */
  dev = (dev_t) rip->i_zone[0];
  put_inode(rip);
  return(dev);
}
```