Link.c

Enlace y desenlace de ficheros

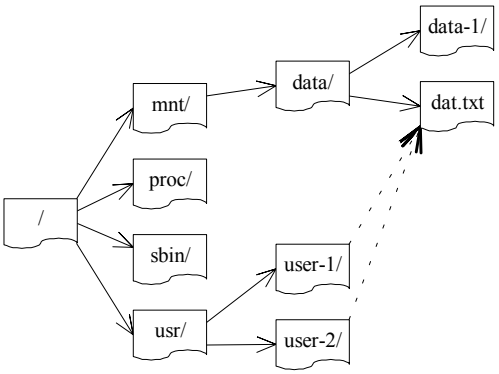
Elia Rivero Díaz
Gabriel Dovalo Carril

Introducción 2
link.c 3
  do_link. 3
  do_unlink 4
  do_rename 5
  Truncate 5
    DO_LINK (código) 7
    DO_UNLINK (código) 8
    DO_RENAME (código) 9
    TRUNCATE (código) 12
    REMOVE_DIR (RM_DIR) (código) 13
    UNLINK_FILE (código) 14
Cuestiones: 16

Introducción

En un sistema en el que se tengan declarados varios usuarios, es necesario permitir la compartición de determinados ficheros si en algún momento varios de estos usuarios deben interactuar como un único grupo de trabajo, por lo que debe existir un método por el cual los ficheros que se consideren necesarios sean vistos, modificados y actualizados para ese grupo de usuarios. Del mismo modo debe existir el método inverso para desvincular a un usuario de los ficheros del grupo a los que antes podía acceder. De estas actividades se encarga el módulo denominado link.c.

Supongamos un fichero (dat.txt) sobre el que se desea que dos usuarios distintos puedan hacer modificaciones, viendo cada uno de ellos las modificaciones del otro.



Este problema se podría solucionar de dos formas distintas:

Archivos de tipo Link: En este caso se crean unos enlaces simbólicos que nos permiten acceder al fichero deseado.

Estructura de datos asociada: Se mantiene una estructura de datos asociada al fichero que contendrá los bloques de disco. Estos bloques serán los i-nodes.

El segundo método es el que utiliza MINIX. En este caso, los directorios ya no contienen bloques de disco, sino punteros a inodes.

Siguiendo esta filosofía, los i-nodes mantendrán información sobre el directorio propietario del archivo y el número de enlaces que están apuntando al archivo en cuestión.

Los directorios, así, se simplifican quedando en forma de archivos con una estructura a base de parejas, en donde el primer campo es un número de i-node y el segundo campo es el nombre de un directorio o de un fichero en código ASCII:

[num. Inode, Nombre ASCII]

Cuando se realiza el proceso de establecer un nuevo enlace a un archivo se crea una nueva entrada en el directorio. El nombre asignado será un nuevo nombre que indique el usuario, mientras que el número de i-node será el mismo que correspondía inicialmente al fichero que queremos enlazar. (Llamada al sistema LINK).

link.c

El archivo link.c se encarga de enlazar y desenlazar ficheros. En este archivo están definidos los siguientes procedimientos:

- do_link
- do_unlink
- do_rename
- truncate
- remove_dir
- unlink_file

A continuación pasamos a describir cada uno de ellos.

do_link.

Función encargada de realizar la llamada al sistema:

```
link(file_name, link_name);
// file_name: nombre del fichero que se desea enlazar.
// link_name: nombre que se le desea asignar al enlace.
```

Esta función realiza una serie de comprobaciones antes de llevar a cabo el enlace de un fichero. Si cualquiera de estas comprobaciones fallase se abortaría la creación de dicho enlace. Cada una de estas salidas por fallo de un chequeo genera un código de error diferente, donde algunos de los posibles errores que pueden ocurrir son:

- *File_name* no existe o es inaccesible.
- *File_name* ya tiene el número máximo de enlaces.
- *File_name* es un directorio y quien realiza la llamada no es el superusuario.
- *Link_name* ya existe.
- *File_name* y *link_name* están sobre dispositivos diferentes.

Si no hay errores presentes, se crea una nueva entrada al directorio con la ristra *link_name* y el número de i-node de *file_name.*

Variables Globales de especial interés:

- *name1* (corresponde a file_name): Nombre del archivo a ser enlazado. Puede tratarse de un fichero o de un subdirectorio
- *name2* (corresponde a link_name): Nombre del subdirectorio en el que se quiere crear el link
- *user_path*: Utilizada por fetch_name para devolver la trayectoria completa hasta el nombre que se le suministra
- *super_user*: Indica si se trata del superusuario

Procedimientos utilizados:

- *fetch_name(nombre, longitud_nombre, flag)*: Dada una trayectoria en el espacio de usuario y su longitud, la copia en user_path.
- *Eat_path(user_path)*: Analizar gramaticalmente la trayectoria dada y localizar su inode en la tabla de inodes.
- *Put_inode(rip)*: Libera un inode. Si nadie más lo está usando, lo escribe a disco. En caso de que este procedimiento descubra que nadie lo está usando y que verdaderamente debe ser liberado, llama a truncate
- *last_dir(path, string)*: Devuelve el inode del último directorio de la trayectoria hasta el fichero a enlazar, colocando el nombre de este fichero a enlazar en string.
- *Advance(ip, string)*: Dado un inode de un directorio y una componente de este directorio, se busca la componente en el directorio, se halla su inode, se abre éste y se devuelve un puntero a su ranura de inode.
- *Search_dir(ip, string, numb, ENTER)*: Crea una entrada en el directorio apuntado por ip con nombre igual a string y inode de dicha entrada igual a numb.

do_unlink

Los archivos y los directorios se eliminan desenlazándolos. El trabajo de las llamadas al sistema UNLINK y RMDIR se hace con *do_unlink*. Nuevamente, se debe hacer una serie de chequeos; la prueba de que un archivo existe y que un directorio no tiene ningún sistema de ficheros montado en él, se hace con el código común en *do_unlink,* y entonces se llama o a *remove_dir* o a *unlink_file*, dependiendo de la llamada al sistema soportada.

Variables de especial interés:

- *name* corresponde a *link_name*

Procedimientos utilizados:

- *search_dir (ip, string, numb, LOOKUP o DELETE):* En el procedimiento anterior se explicó el caso en el cual era ENTER. Si es LOOKUP busca la *string* en el directorio del inode apuntado por *ip*, y devuelve el inode que le corresponde. Si es DELETE elimina la *string* del directorio.

do_rename

La otra llamada al sistema soportada en *link.c* es RENAME. Los usuarios de UNIX están familiarizados con el comando shell *mv* que finalmente usa esta llamada; su nombre refleja otro aspecto de la llamada. No sólo puede cambiar el nombre de un archivo dentro de un directorio, sino que puede también mover el archivo desde un directorio a otro, y puede hacer esto atómicamente, lo cual evita problemas. Esta tarea la realiza *do_rename*.

Hay muchas condiciones que deben probarse antes de que pueda completarse este comando, algunas de las cuales son:

1) El archivo original debe existir.
2) El antiguo pathname no debe ser un prefijo del nuevo pathname en el árbol directorio, es decir, no es posible una llamada de la forma:
   rename("/usr/tmp","/usr/tmp/lib")
3) No se acepta ni "." ni ".." como antiguo o nuevo nombre.
4) Ambos directorios padre deben estar sobre el mismo dispositivo.
5) Ambos directorios padre deben estar sobre un dispositivo accesible, y además deben ser accesibles.
6) Ni el antiguo ni el nuevo nombre puede ser un directorio con un sistema de archivos montado sobre él.
7) Debe ser posible eliminar un archivo ya existente si se va a crear uno nuevo con ese mismo nombre. Si el nuevo archivo va a estar en el mismo directorio que el antiguo borramos la entrada del antiguo y creamos la nueva en su lugar. En caso contrario (los ficheros están en diferentes directorios) creamos primero la nueva entrada y después borramos la vieja.

Truncate

Podría suceder que un archivo dado haya eliminado todos sus enlaces con directorios, con lo cual ya no habría nadie que pudiese acceder a él. (Debemos recordar que un campo en el inode del fichero controla el numero de enlaces del fichero). En el caso de que el numero de enlaces de otros directorios con este fichero sea cero, tanto el archivo como toda la información asociada deberán ser eliminados. Esta es la función de **truncate**.

Un pequeño esbozo de la tarea de truncate podría ser la siguiente:

- Calcular el tamaño de la zona.
- En caso de existir asignar el tamaño de los pipes.
- Liberar las zonas de datos.
- Liberar las zonas indirectas (sencillas y dobles).

Variables locales:

- *Zone_size:* Tamaño de la zona en bytes.
- *Scale:* Número de bloques por zona.

Procedimientos utilizados:

- *Scale_factor (rip):* Devuelve el factor de escala para convertir bloques en zonas.
- *Read_map(rip, position):* Dado un inode y una posición dentro del archivo correspondiente, localiza el número del bloque (no de zona) en el que se hallará esa posición y lo devuelve.
- *Free_zone (dev, z):* Libera una zona de memoria asignada a un fichero.
- *Get_block (dev, b, NORMAL):* Pide un bloque para lectura o escritura.
- *Put_block (bp, INDIRECT_BLOCK):* Libera el bloque pedido.
- *Wipe_inode (rip):*Trunca un inode existente.

```
/******* CODIGO *******/

/* Este fichero maneja las llamadas al sistema "LINK" y "UNLINK".
Además se ocupa de la liberación del espacio utilizado por un fichero
cuando se efectuó el último UNLINK sobre dicho fichero, en cuyo caso
los bloques ocupados deben ser devueltos a la lista de bloques libres.
 *
 * Los puntos de acceso a este fichero son:
 *   do_link:     realiza la llamada al sistema "LINK".
 *   do_unlink:   realiza las llamadas al sistema "UNLINK" y "RMDIR".
 *   do_rename:   realiza la llamada al sistema "RENAME"
 *   truncate:    libera todos los bloques asociados a un i-node.
 */

#include "fs.h"
#include <sys/stat.h>
#include <string.h>
#include <minix/callnr.h>
#include "buf.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "param.h"
#include "super.h"

#define SAME 1000

FORWARD _PROTOTYPE( int remove_dir, (struct inode *rldirp, struct
inode *rip, char dir_name[NAME_MAX]));

FORWARD _PROTOTYPE( int unlink_file, (struct inode *dirp,  struct
inode *rip, char file_name[NAME_MAX]));

/*****************************************************************/
                          DO_LINK (código)
/*****************************************************************/

PUBLIC int do_link()
{
/* Realiza la llamada al sistema "link(name1, name2)"            */
/*    name1: fichero que se desea enlazar                        */
/*    name2: nombre que se desea dar al fichero una vez enlazado */

  register struct inode *ip, *rip;
  register int r;
  char string[NAME_MAX];
  struct inode *new_ip;

  /* Comprueba si 'name1' (archivo a ser enlazado) existe */
  /* busca el fichero a enlazar e introduce su ruta en el área de
usuario */
  if (fetch_name(name1, name1_length, M1) != OK) return(err_code);
  /* ahora intenta acceder a ese fichero, si no puede acceder a su i-
node se produce un error */
  if ((rip = eat_path(user_path)) == NIL_INODE) return(err_code);

  /* Verifica si el fichero ya tiene el máximo número de enlaces */
  r = OK;
  if ((rip->i_nlinks & BYTE) >= LINK_MAX) r = EMLINK;

  /* Sólo el superusuario puede enlazar directorios */
```

```
  if (r == OK)
      if ((rip->i_mode & I_TYPE) == I_DIRECTORY && !super_user)
            r = EPERM;
/* Si hay algún error relacionado con 'name1', liberar el i-node. */
  if (r != OK) {
      put_inode(rip);
      return(r);
  }
/* ¿Existe el directorio final 'name2'? */
  if (fetch_name(name2, name2_length, M1) != OK) {
      put_inode(rip);
      return(err_code);
  }
  if ((ip = last_dir(user_path, string)) == NIL_INODE) r = err_code;

/* comprobamos si el i-node obtenido en la línea anterior es un
directorio, a un fichero, u otra cosa... */
if (r == OK) {
      if ((new_ip = advance(ip, string)) == NIL_INODE) {
            r = err_code;
            if (r == ENOENT) r = OK; /* no existe tal directorio */
      } else {
            put_inode(new_ip);
            r = EEXIST;} }
/* Comprobar si existen enlaces a través de dispositivos */
/* no se deben permitir enlaces a sistemas de fichero diferentes */
  if (r == OK)
      if (rip->i_dev != ip->i_dev) r = EXDEV;

/* Intenta enlazar, crea un i-node asociado a "string" */
  if (r == OK)
      r = search_dir(ip, string, &rip->i_num, ENTER);

/* Si todo ha ido bien, registrar el enlace */
  if (r == OK) {
      rip->i_nlinks++;
      rip->i_update |= CTIME;
      rip->i_dirt = DIRTY;
  }

/* Hecho.  Liberar ambos i-nodes. */
  put_inode(rip);
  put_inode(ip);
  return(r);
}

/*****************************************************************/
                          DO_UNLINK (código)
/*****************************************************************/


PUBLIC int do_unlink()
{
/* Realiza la llamada al sistema "unlink(name)" o "rmdir(name)". El
código para ambas llamadas es prácticamente idéntico.  Sólo difieren
en algunas condiciones de chequeo.  "Unlink()" puede usarse por el
superusuario para hacer determinadas operaciones peligrosas; "rmdir()"
no. */

  register struct inode *rip;
  struct inode *rldirp;
```

```
  int r;
  char string[NAME_MAX];

  /* Tomamos el último directorio de la ruta */
  if (fetch_name(name, name_length, M3) != OK) return(err_code);
  if ((rldirp = last_dir(user_path, string)) == NIL_INODE)
      return(err_code);

/* El último directorio existe.  ¿Existe el fichero también? */
  r = OK;
  if ((rip = advance(rldirp, string)) == NIL_INODE) r = err_code;

/* Si hay error, devolver el i-node */
  if (r != OK) {
      put_inode(rldirp);
      return(r);
  }

/* No se puede borrar un lugar donde se haya realizado un mount, ya
que no se permite borrar un punto de anclaje de un sistema de
fihceros. */
  if (rip->i_num == ROOT_INODE) {
      put_inode(rldirp);
      put_inode(rip);
      return(EBUSY);
  }

/* Ahora chequeamos si la llamada es permitida, separadamente para
"unlink()" y "rmdir()" */
  if (fs_call == UNLINK) {
      /* Sólo el Superusuario puede desenlazar directorios, pero el
superusuario puede desenlazar cualquier directorio.*/
      if ( (rip->i_mode & I_TYPE) == I_DIRECTORY && !super_user) r =
EPERM;

      /* No se puede desenlazar un fichero si es la raíz de un sistema
de ficheros. */
      if (rip->i_num == ROOT_INODE) r = EBUSY;

/* Realmente intenta desenlazar el fichero; falla si su padre está en
modo 0 etc. */
      if (r == OK) r = unlink_file(rldirp, rip, string);

  } else {
      r = remove_dir(rldirp, rip, string); /* La llamada al sistema es
RMDIR */
  }

  /* Si el desenlace fue posible, ya ha sido realizado, en otro caso
no. */
  put_inode(rip);
  put_inode(rldirp);
  return(r);
}

/*************************************************************************/
                              DO_RENAME (código)
/*************************************************************************/

PUBLIC int do_rename() /* Realiza la llamada al sistema rename(name1,
name2) en donde se cambia el nombre del enlace 'name1' a 'name2'. */
```

```
{
struct inode *old_dirp, *old_ip;
/* punteros a los inodes del directorio donde se encuentra el fichero
a renombrar y del propio fichero a renombrar*/
struct inode *new_dirp, *new_ip;    /* punteros a los inodes del nuevo
directorio y del  nuevo fichero.*/
  struct inode *new_superdirp, *next_new_superdirp;
  int r = OK;     /* Flag de error, inicialmente sin error */
  int odir, ndir;/* TRUE si los ficheros {viejo|nuevo} son dirs*/
  int same_pdir;  /* TRUE si los directorios padre son el mismo */
  char old_name[NAME_MAX], new_name[NAME_MAX];
  ino_t numb;
  int r1;

  /* Mira si 'name1' (fichero a renombrar) existe.  Toma los inodes
del directorio y del fichero. */
  if (fetch_name(name1, name1_length, M1) != OK) return(err_code);
  if ( (old_dirp = last_dir(user_path, old_name))==NIL_INODE)
      return(err_code);
  if ( (old_ip = advance(old_dirp, old_name)) == NIL_INODE)
      r = err_code;
/* Mira si 'name2' (nuevo nombre) existe. Toma los inodes del
directorio y del fichero. */
  if (fetch_name(name2, name2_length, M1) != OK) r = err_code;
  if ( (new_dirp = last_dir(user_path, new_name)) == NIL_INODE)
      r = err_code;
  new_ip = advance(new_dirp, new_name);
/* no se requiere su existencia. */
  if (old_ip != NIL_INODE)
      odir = ((old_ip->i_mode & I_TYPE) == I_DIRECTORY);
      /* TRUE si es un directorio. */

/* Si todo va bien, comprobar varios posibles errores. */
  if (r == OK) {
      same_pdir = (old_dirp == new_dirp);

      /* El inode antiguo no debe estar en la ruta al directorio
definitivo. */
      if (odir && !same_pdir) {
            dup_inode(new_superdirp = new_dirp);
            /* Realizamos un bucle para descender por la cadena de
nodos-i a partir de una copia del nuevo directorio a crear, para ver
si ocasiona algún tipo de problemas */
            while (TRUE)
            {/* puede quedarse en un bucle del sistema de ficheros. */
                  if (new_superdirp == old_ip) {
                        r = EINVAL;
                        break;      }
      next_new_superdirp = advance(new_superdirp, dot2);
                  put_inode(new_superdirp);
                  if (next_new_superdirp == new_superdirp)
                        break;
/* vuelta al directorio raíz del sistema. */
                  new_superdirp = next_new_superdirp;
                  if (new_superdirp == NIL_INODE) {
                  /* Se ha perdido la entrada "..".Asumimos lo peor.*/
                        r = EINVAL;
                        break;
                  }
            }
            put_inode(new_superdirp);
```

```
      }

      /* El nombre antiguo o nuevo no deben ser . o .. */
      if (strcmp(old_name, ".")==0 || strcmp(old_name, "..")==0 ||
          strcmp(new_name, ".")==0 || strcmp(new_name, "..")==0)
r = EINVAL;

      /* Ambos directorios padre deben estar sobre el mismo tipo de
sistema de ficheros. (Vease 'man link') */
      if (old_dirp->i_dev != new_dirp->i_dev) r = EXDEV;
      /* Debemos poder escribir en los directorios padre, poder buscar
en ellos y deben estar en un dispositivo en el que podamos escribir.
*/
      if ((r1 = forbidden(old_dirp, W_BIT | X_BIT)) != OK ||
          (r1 = forbidden(new_dirp, W_BIT | X_BIT)) != OK) r = r1;

      /*Algunos chequeos se realizan solo si existe la nueva ruta.*/
      if (new_ip == NIL_INODE) {
            /* No podemos renombrar un fichero donde hayamos montado un
            sistema de ficheros. */
            if (old_ip->i_dev != old_dirp->i_dev) r = EXDEV;
            /* Además comprobamos el no haber sobrepasado el número
máximo de enlaces */
            if (odir && (new_dirp->i_nlinks & BYTE) >= LINK_MAX &&
                !same_pdir && r == OK) r = EMLINK;
      } else {
            if (old_ip == new_ip) r = SAME; /* old=new */
/* tienen el viejo o el nuevo fichero un sistema de ficheros montado
en ellos? */
            if (old_ip->i_dev != new_ip->i_dev) r = EXDEV;
            ndir = ((new_ip->i_mode & I_TYPE) == I_DIRECTORY);
            if (odir == TRUE && ndir == FALSE) r = ENOTDIR;
            if (odir == FALSE && ndir == TRUE) r = EISDIR;
      }
}
/* Si un proceso tiene un directorio raíz que no es la raíz del
sistema, podríamos estar moviendo "accidentalmente" su directorio de
trabajo a un lugar donde su directorio raíz no sea un superdirectorio.
Esto puede hacer que la función chroot pierda su funcionalidad. Si
chroot se usara frecuentemente, deberíamos chequearlo probablemente
aquí. */

  /* El cambio de nombre probablemente irá bien. En este punto sólo
nos podrían fallar dos cosas:
   * 1. No podemos borrar el nuevo fichero (Cuando el nuevo fichero ya
existe).
   * 2. No podemos crear la nueva entrada de directorio. (el nuevo
fichero NO existe)
   *     [El directorio debe crecer un bloque y no puede porque el
disco está lleno. */
  if (r == OK) {
      if (new_ip != NIL_INODE) {
             /* Ya existe una entrada 'new', Intentemos borrarla. */
            if (odir)
                  r = remove_dir(new_dirp, new_ip, new_name);
            else
                  r = unlink_file(new_dirp, new_ip, new_name);
      }
      /* Si  r = OK, el cambio de nombre se realizará, mientras haya
una entrada libre en el nuevo directorio padre.*/
  }
```

```
if (r == OK) {
/* Si el nuevo nombre va a estar en el mismo directorio padre que el
viejo, primero borraremos el viejo nombre con el fin de liberar una
entrada para el nuevo. En otro caso, primero intentaremos crear la
entrada del nuevo nombre para asegurar el éxito en el cambio de nombre
*/
      numb = old_ip->i_num;   /* Número de inode del fichero antiguo */
      if (same_pdir) {
            r = search_dir(old_dirp, old_name, (ino_t *) 0, DELETE);
                                    /* No debería ir mal. */
            if (r==OK) (void) search_dir(old_dirp, new_name, &numb,
ENTER);
      } else {
            r = search_dir(new_dirp, new_name, &numb, ENTER);
            if (r == OK)
                (void) search_dir(old_dirp, old_name, (ino_t *) 0,
DELETE);
      }
  }
/* Si r= OK los atributos ctime y mtime del viejo directorio habrán
sido marcados para su posterior actualización en search_dir. */

  if (r == OK && odir && !same_pdir) {
/* Actualiza la entrada ".." en el directorio (todavía apuntaba al
old_dirp) */
      numb = new_dirp->i_num;
      (void) unlink_file(old_ip, NIL_INODE, dot2);
      if (search_dir(old_ip, dot2, &numb, ENTER) == OK) {
            /* Nuevo link creado. */
            new_dirp->i_nlinks++;
            new_dirp->i_dirt = DIRTY;
      }
  }

  /* Liberamos los inodes. */
  put_inode(old_dirp);
  put_inode(old_ip);
  put_inode(new_dirp);
  put_inode(new_ip);
  return(r == SAME ? OK : r);
}

/*****************************************************************/
                          TRUNCATE (código)
/*****************************************************************/

/* Si todos los enlaces a un i-node dado han sido borrados se deberán
borrar también todos los bloques que cuelgan de dicho i-node, y
liberarlo a él mismo. */

PUBLIC void truncate(rip)
register struct inode *rip;
/* Puntero al inode que ha de ser truncado. */
{
/* Borrar todas las zonas del inode "rip" y marcarlas a dirty.*/

  register block_t b;
  zone_t z, zone_size, z1;
  off_t position;
  int i, scale, file_type, waspipe, single, nr_indirects;
  struct buf *bp;
```

```
  dev_t dev;

  file_type = rip->i_mode & I_TYPE;   /* Comprobemos si el fichero es
especial.*/
  if (file_type == I_CHAR_SPECIAL || file_type == I_BLOCK_SPECIAL)
return;
  dev = rip->i_dev;           /* dispositivo donde está el inode. */
  scale = rip->i_sp->s_log_zone_size;
  zone_size = (zone_t) BLOCK_SIZE << scale;
  nr_indirects = rip->i_nindirs;
/* Los canales pueden acortarse en tamaño, así que ajustemos el tamaño
para asegurar que se borran todas las zonas. */
  waspipe = rip->i_pipe == I_PIPE;  /* TRUE si era una tubería */
  if (waspipe) rip->i_size = PIPE_SIZE;

  /* Recorre el fichero de zona en zona, localizando y liberándolas */
  for (position = 0; position < rip->i_size; position += zone_size) {
      if ( (b = read_map(rip, position)) != NO_BLOCK) {
            z = (zone_t) b >> scale;
            free_zone(dev, z);
      }
  }

/* Ya se han liberado todas las zonas de datos. Ahora liberaremos las
zonas indirectas. */
  rip->i_dirt = DIRTY;
  if (waspipe) {
      wipe_inode(rip);  /* Se limpia el inode de tuberías */
      return;  /* los bloques indirectos contienen posiciones de
ficheros*/
  }
  single = rip->i_ndzones;
  free_zone(dev, rip->i_zone[single]);  /* Zonas simples indirectas */
if ( (z = rip->i_zone[single+1]) != NO_ZONE) {
/*Libera todas las zonas simples indirectas apuntadas por i_zone[]. */
      b = (block_t) z << scale;
      bp = get_block(dev, b, NORMAL);
      /*Obtiene la zona doble indirecta.*/
      for (i = 0; i < nr_indirects; i++) {
            z1 = rd_indir(bp, i);
            free_zone(dev, z1);
      }

      /* Ahora libera la zona doble indirecta en sí. */
      put_block(bp, INDIRECT_BLOCK);
      free_zone(dev, z);
  }

  /* Deja los número de zona para de(1) para recuperar el fichero
despues de un unlink(2).  */
}

/*****************************************************************/
                      REMOVE_DIR (RM_DIR) (código)
/*****************************************************************/


PRIVATE int remove_dir(rldirp, rip, dir_name)
struct inode *rldirp;               /* directorio padre */
struct inode *rip;                  /* directorio a borrar */
```

```
char dir_name[NAME_MAX];           /* Nombre del directorio a borrar.
*/
{
  /* Se procede a borrar un directorio y tienen que darse cinco
condiciones:
   *  - La entrada a borrar tiene que ser un directorio.
   *  - El directorio tiene que estar vacio (excepto el  . y ..)
   *  - La última componente del camino no puede ser . o ..
   *  - El directorio no puede ser la raíz de un sistema de ficheros
montado.
   *  - El directorio no puede ser ni el directorio de trabajo ni el
directorio raíz de nadie.
   */

  int r;
  register struct fproc *rfp;
/* search_dir comprueba que rip es también un directorio. */
  if ((r = search_dir(rip, "", (ino_t *) 0, IS_EMPTY)) != OK)
return r;

/* No se puede borrar el directorio '.' ni el '..' */
  if (strcmp(dir_name, ".") == 0 || strcmp(dir_name, "..") == 0)
return(EINVAL);
  if (rip->i_num == ROOT_INODE) return(EBUSY);
/* No podemos borrar 'root' */

/* Nos aseguramos de no borrar el directorio de trabajo de nadie */
  for (rfp = &fproc[INIT_PROC_NR + 1]; rfp < &fproc[NR_PROCS]; rfp++)
      if (rfp->fp_workdir == rip || rfp->fp_rootdir == rip)
            return(EBUSY);

  /* Ahora intenta desenlazar el fichero; falla si su padre está en
modo 0 etc. */
  if ((r = unlink_file(rldirp, rip, dir_name)) != OK) return r;

 /* Desenlaza . y .. Del directorio. El superusuario puede enlazar y
desenlazar cualquier directorio, así que no debemos asumir nada sobre
ellos.*/
  (void) unlink_file(rip, NIL_INODE, dot1);
  (void) unlink_file(rip, NIL_INODE, dot2);
  return(OK);
}

/*****************************************************************/
                          UNLINK_FILE (código)
/*****************************************************************/


PRIVATE int unlink_file(dirp, rip, file_name)
struct inode *dirp;          /* Directorio padre del fichero */
struct inode *rip; /*inode del fichero, puede ser NIL_INODE también.*/
char file_name[NAME_MAX];    /* Nombre del fichero a borrar. */
{
/* Desenlazamos 'file_name'; rip deberá ser el i-node de 'file_name' o
NIL_INODE. */

  ino_t numb;                /* Número de inodo */
  int r;

 /* Si rip no es NIL_INODE, se usa para tener un acceso más rápido al
i-node. */
```

```
  if (rip == NIL_INODE) {
/* Buscamos el  fichero en un directorio con el fin de intentar
obtener su i-node. */
      err_code = search_dir(dirp, file_name, &numb, LOOK_UP);
      if (err_code == OK) rip = get_inode(dirp->i_dev, (int) numb);
      if (err_code != OK || rip == NIL_INODE) return(err_code);
  } else {
      dup_inode(rip);   /* El i-node será devuelto por put_inode */
  }
  r = search_dir(dirp, file_name, (ino_t *) 0, DELETE);

  if (r == OK) {
      rip->i_nlinks--;  /* Entrada borrada desde el directorio padre */
      rip->i_update |= CTIME;
      rip->i_dirt = DIRTY;
  }
  put_inode(rip);
  return(r);
}
```

Cuestiones:

1) ¿Qué pasos realiza la llamada al sistema 'rmdir'?

- Lo primero es comprobar que el nombre especificado existe y que es un directorio.
- También se comprueba que dicho directorio no se corresponde con las entradas "." y ".." ya que son entradas especiales que no pueden ser borradas.
- No pueden ser borrados ni el directorio raíz ni ningún directorio asignado previamente como directorio de usuario.
- En caso de que todas las condiciones se cumplan, se hace una llamada a unlink_file que se encargará de borrar la entrada del directorio indicado.
- Una vez hecho esto, llamamos dos veces a la función unlink_file pasándole como directorios a borrar los "." y ".." para eliminar definitivamente el directorio.

2) ¿En que consiste la función unlink_file?

Es la función que se encarga de la eliminación de un enlace a un fichero o directorio.
Básicamente tiene los siguientes pasos sabiendo que file_name es el nombre del fichero o directorio a eliminar:

- Obtenemos el i-node correspondiente al fichero al que le sacaremos un enlace mediante el uso de las funciones search_dir y get_inode .
- Si no era el último enlace, tomamos el inode y actualizamos su número de enlaces, de lo contrario el i-node se libera.

3) ¿Cómo se llevan a cabo el cambio de nombre de un link o fichero?

Suponiendo que la llamada se hace de la forma do_rename (nombre1, nombre2) siendo nombre1 el nombre original y nombre2 el nuevo nombre, los pasos serían:

- Comprobar que el fichero a renombrar existe.
- Comprobar si el nuevo nombre existe.
- Para realizar el cambio, el nombre antiguo no debe ser un superdirectorio del nuevo o sea, que no podemos hacer una llamada de la forma rename("/usr/tmp","/usr/tmp/lib").
- Comprobar que ninguno de los nombres debe ser "." ó "..".
- Comprobar que ambos directorios padre deben estar sobre el mismo dispositivo.
- Se deberá poder escribir en los directorios padre, poder buscar en ellos y deberán estar en un dispositivo en el que se pueda escribir.
- No se puede renombrar un fichero donde esté montado un sistema de ficheros.
- En este punto se deberá tener en cuenta que si ya existe un fichero con el nuevo nombre, éste será borrado por lo que si no se tuviesen los permisos adecuados para borrarlo, la operación fracasaría. También puede suceder que no quede espacio para ubicar la nueva entrada.
- Finalmente se lleva a cabo el cambio de nombre, teniendo en cuenta que si el nuevo nombre va a estar en el mismo directorio padre que el viejo, primero se borrará el nombre más viejo para liberar una entrada para utilizar con el nuevo. En otro caso, primero se intentaría crear la entrada del nuevo nombre para asegurar el éxito en el cambio de nombre.