# Ampliación Sistemas Operativos

# Protect.c

Patricia Afonso Godoy
María Teresa Cardona Perdomo

© Universidad de Las Palmas de Gran Canaria

# INDICE

**1.- DESCRIPCIÓN FUNCIONAL........................................................ 2**

**2.- MECANISMO DE PROTECCIÓN ................................................. 3**
- ♦ Llamada al sistema chmod. ...................................................... 3
- ♦ Llamada al sistema chown........................................................ 3
- ♦ Llamada al sistema umask. ...................................................... 3
- ♦ Llamada al sistema Access. ...................................................... 4
- ♦ Función Forbidden................................................................. 4
- ♦ Función Read_only. ................................................................ 4

**3.- LLAMADAS DEL SISTEMA.......................................................... 5**

**4.- ESTRUCTURAS UTILIZADAS..................................................... 8**

**5.- FUNCIONES EXTERNAS............................................................ 8**

**6.- CÓDIGO FUENTE....................................................................... 9**

6.1.- do_chmod .......................................................................... 9
6.2.- do_chown .......................................................................... 12
6.3.- do_mask ............................................................................ 14
6.4.- do_access .......................................................................... 15
6.5.- Forbidden: .......................................................................... 16
6.6.- Read_only:.......................................................................... 20

**7.- PREGUNTAS:............................................................................ 21**

7.1.- Hacer una traza desde que un usuario hace un chmod hasta que se ejecuta la función do_chmod.......................................... 21
7.2.- ¿Qué hace y cómo funciona la función do_chmod?......................... 21
7.3.- ¿Qué hace y cómo funciona la función do_chown?......................... 21
7.4.- ¿Qué hace y cómo funciona la función do_umask?......................... 21
7.5.- ¿Qué hace y cómo funciona la función do_access?......................... 21
7.6.- ¿Qué hace y cómo funciona la función forbidden?.......................... 21

## LLAMADAS AL SISTEMA PARA PROTECCIÓN

# 1.- DESCRIPCIÓN FUNCIONAL

Los archivos en MINIX se valen de 11 bits para describir el modo de protección.

Los nueve bits menos significativos se utilizan para los permisos de lectura, escritura y ejecución, para el propietario, el grupo y el resto.

Los 2 bits más significativos de protección corresponden al SETGID (establecer el identificador del grupo) y al SETUID (establecer el identificador del usuario). Estos bits se usan para determinar si se adquieren los identificadores efectivos del fichero accedido, tanto el del usuario como el del grupo. Cuando algún usuario ejecuta un programa con el bit SETUID activado el uid efectivo del usuario se cambiará por el UID del propietario del fichero durante la ejecución del programa. La utilidad más frecuente de estos bits estriba en permitir a los usuarios ejecutar programas que utilizan funciones de superusuario (como puede ser la creación de directorios).

Debido a esta funcionalidad, se debe diferenciar los identificadores efectivos y los reales. Cuando se ejecuta un archivo que tenga el bit SETUID o SETGID activado, el identificador real de usuario o grupo será cambiado por el identificador efectivo en ese momento del usuario o grupo. Así, el identificador efectivo del archivo sólo será válido durante la ejecución del mismo, después permanecerá los identificadores reales.

| | | PROPIETARIO | | | GRUPO | | | OTROS | | |
|---|---|---|---|---|---|---|---|---|---|---|
| SETUID | SETGID | R | W | X | R | W | X | R | W | X |

Para averiguar los uid o gid efectivos y reales se usan las llamadas al sistema **GETUID** y **GETGID**.

El superusuario puede cambiar los identificadores de usuario y de grupo con la llamada al sistema **SETUID** y **SETGID**, y también cambiar el propietario del archivo.

# 2.- MECANISMO DE PROTECCIÓN

En la tabla i-node del fichero además de las direcciones de sus bloques de disco, se encuentran los atributos del fichero entre los cuales se distinguen los UID y GID del fichero (campos i_uid e i_gid de struct inode) y las protecciones (campo i_mode de struct inode).

El mecanismo de protección en Minix utiliza los bits de lectura-escritura-ejecución o *rwx*.

## ♦ Llamada al sistema chmod.

La llamada al sistema **chmod** permite establecer y cambiar el modo de acceso de un archivo. Por ejemplo la siguiente llamada sólo permite la ejecución al propietario del archivo:

chmod("archivo",0766)

Esta llamada corresponde a la función *do_chmod* en el archivo protec.c. Después de efectuar una serie de verificaciones de validez, el modo se cambia.

## ♦ Llamada al sistema chown.

El superusuario cuenta con la posibilidad de cambiar el propietario o el grupo de un archivo, para ello cuenta con la llamada al sistema **chown**. Un usuario normal sólo puede, mediante esta llamada al sistema, cambiar el grupo de sus propios ficheros.

Tanto esta función como la anterior modifican un campo del nodo-i del fichero, siendo su implementación similar.

Esta llamada corresponde a la función *do_chown* en el archivo protect.c.

## ♦ Llamada al sistema umask.

La llamada al sistema **umask** permite al usuario fijar una máscara de limitación de acceso a los ficheros (almacenada en la tabla de procesos), que se usará para enmascarar los bits del modo cuando se crea un archivo (*creat*).

Por ejemplo, después de una llamada al sistema *umask(022),* si se realiza una llamada del tipo *creat("archivo",0777)*, asignará 0755 al modo en lugar de 0777. Puesto que la máscara de bits es heredada por los procesos hijos, si el shell (intérprete de comandos) ejecuta un *umask* justo después del inicio de sesión, ninguno de los procesos de usuario de esa sesión creará accidentalmente archivos en los que otras personas puedan escribir.
Esta llamada corresponde a la función *do_umask* en el archivo protec.c. Esta función contendría una sola línea si no tuviera que devolver el valor antiguo de la máscara como resultado.

## ♦ Llamada al sistema Access.

La llamada al sistema **access** averigua si un usuario tiene acceso a un determinado fichero (por ejemplo para lectura),es decir, comprobar si el acceso está permitido para el identificador de usuario real (uid).

Se implementa mediante el procedimiento **do_access** que toma el i-node del fichero y llama al procedimiento interno **forbidden**, para ver si el acceso al fichero está prohibido.

## ♦ Función Forbidden.

**Forbidden** es un procedimiento interno que da la posibilidad de comprobar si para un i-node dado se permite un modo de acceso deseado.

**Forbidden** comprueba el uid y el gid, así como la información contenida en el i-node del fichero. Según lo que encuentre, selecciona uno de los tres grupos de bits rwx y comprueba si el acceso se permite o no.

## ♦ Función Read_only.

Read_only es un procedimiento interno que tiene como parámetro de entrada el i-node, para indicar si el sistema de archivos en el que está situado el i-node está montado sólo para lectura o para lectura y escritura. Esto es necesario para evitar la escritura en sistemas de archivos que se montaron solo para lectura.

# 3.- LLAMADAS DEL SISTEMA

El usuario hace una llamada al sistema como access(), chmod(), chown(), umask(). Cada una de estas llamadas va a una librería en *src/lib/posix/,* se prepara un mensaje con los parámetros de entrada y se realiza una llamada _syscall que genera la interrupción 33 y esta provoca la llamada al *sys*_call de mpx.

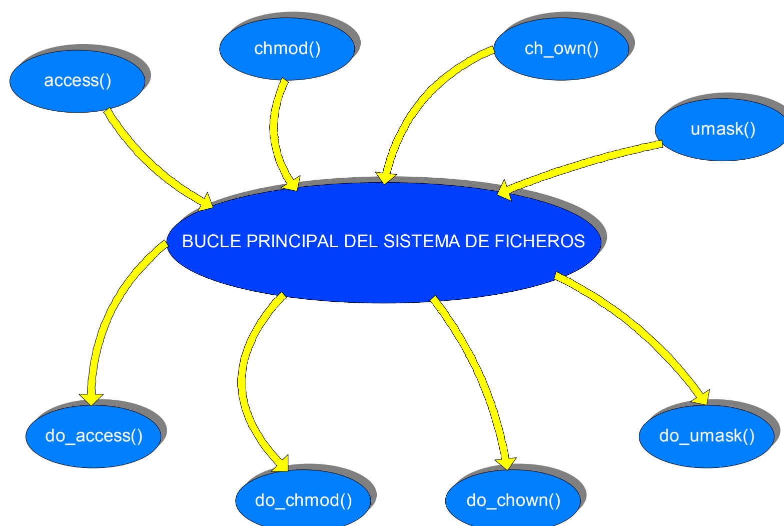
Cuando un programa de usuario ejecuta la instrucción:

n=chmod (name,mode)

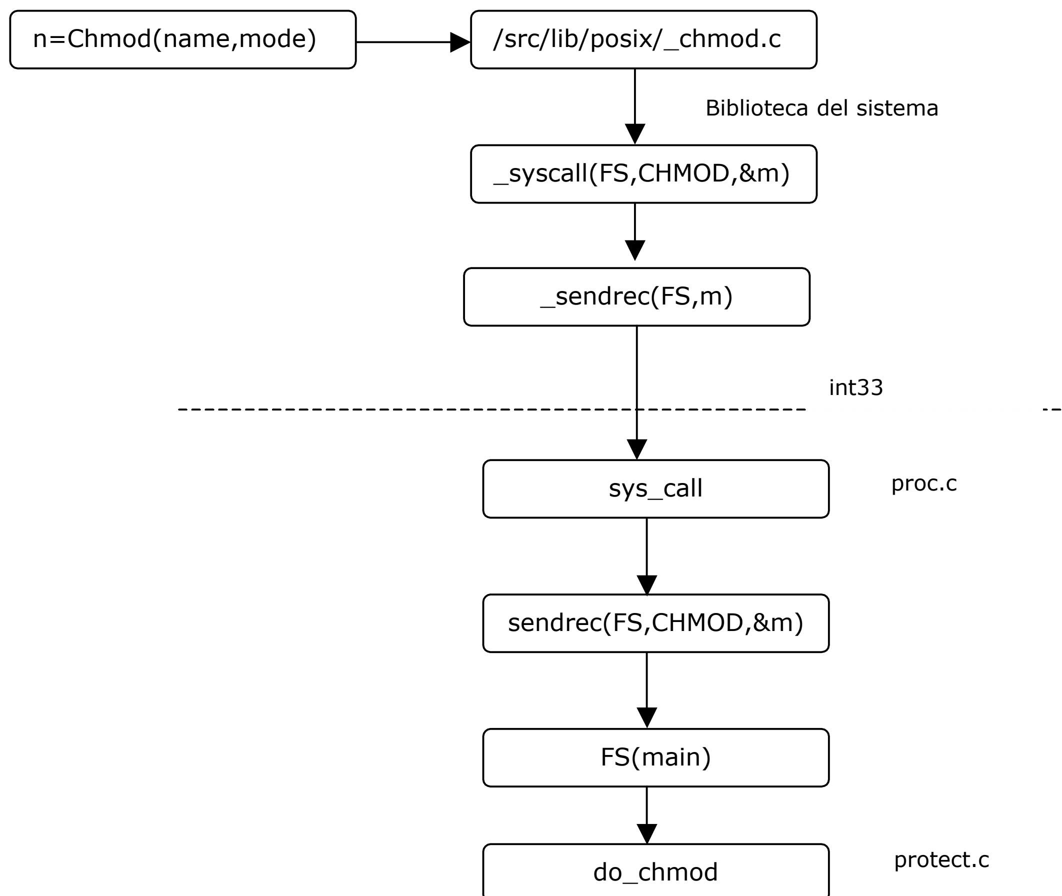
para cambiar los permisos de un archivo, se invoca al procedimiento de biblioteca chmod() con estos 2 parámetros. Ahora este procedimiento construye la estructura del mensaje con los 2 parámetros y hace la llamada _syscall con destino FS e indicando operación chmod. A continuación _syscall hace una llamada a *sendrec* que hace la interrupción software int33, que después de unos pasos se llega a sys_call de mpx.

Ahora en sys_call() que está en proc.c se construye un mensaje con destino FS (mini_rec), quedando bloqueado en espera de respuesta.

El **main** del FS usa el tipo de mensaje como índice en el vector **call_vec** para llamar el procedimiento que se encarga de la operación de cambio de permisos do_chmod.



```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    src/lib/posix/_chmod.c
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

39300   #include <lib.h>

39301   #define chmod   _chmod

39302   #include <sys/stat.h>

39303
39304   PUBLIC int chmod(name, mode)

39305   _CONST char *name;

39306   Mode_t mode;

39307   {

39308     message m;
39309

39310     m.m3_i2 = mode;

39311     _loadname(name, &m);  ¿?? que haces

39312     return(_syscall(FS, CHMOD, &m));

39313   }
```


n=Chmod(name,mode)
/src/lib/posix/_chmod.c
Biblioteca del sistema
_syscall(FS,CHMOD,&m)
_sendrec(FS,m)
int33
sys_call
proc.c
sendrec(FS,CHMOD,&m)
FS(main)
do_chmod
protect.c

# 4.- ESTRUCTURAS UTILIZADAS

a) Estructura inode (fichero inode.c):

- **i_mode**: modo de acceso.
- **i_dev:** dispositivo sobre el que se encuentra el nodo i.
- **i_dirt:** indica si el archivo es DIRTY.
- **i_gid** y **i_uid:** identificadores de grupo y usuario, respectivamente.
- **i_update:** Almacena los bits de ATIME, CTIME y MTIME.
- **i_mount:** este bit se activa si el sistema de ficheros está montado.
- **i_sp:** puntero al superbloque del dispositivo del i-node.

**b)** Estructura fproc "tabla de procesos" (fichero fproc.h):

- **fp_effuid, fp_effgid**: identificadores de grupo y usuario efectivos para el proceso.
- **fp_umask:** máscara de protección para los ficheros creados por el proceso**.**
- **fp_realuid** y **fp_realgid:** Identificadores de grupo y usuario reales para el proceso.

**c)** Estructura super_block (fichero super.c) "El superbloque contiene información sobre el sistema de ficheros".

- **s_rd_only:** indica si el sistema de archivos montado es de sólo lectura.
- **s_imount:** nodo-i raiz del sistema de archivo montado.
- **s_dev:** dispositivo del cual provino el superbloque del sistema de archivo montado.

# 5.- FUNCIONES EXTERNAS

- ***fetch_name (path,len,flag)***: toma la trayectoria y la copia en **user_path [UTILITY.C]**.

- ***eat_path (user_path):*** captura el nodo i para **user_path [PATH.C]**.

- ***put_inode (rip)***: devuelve el i-node a la tabla de i-nodes una vez que el procedimiento ha acabado con él **[INODE.C]**.

- **get_inode (dev,numb),** busca en la tabla de i-nodes uno dado. **dev** es el dispositivo donde reside el i-node y **numb** es el número del i-node **[INODE.C]**.

- **get_super (dev)**, busca la entrada al superbloque con el dispositivo **dev**. **[SUPER.C]**.

# 6.- CÓDIGO FUENTE

Este fichero se encarga de la protección en el sistema de ficheros. Contiene el código de cuatro llamadas al sistema relativas a la protección.

Los puntos de entrada son:

- do_chmod: realiza la llamada a sistema CHMOD.
- do_chown: realiza la llamada a sistema CHOWN.
- do_umask: realiza la llamada a sistema UMASK.
- do_access: realiza la llamada a sistema ACCESS.
- forbidden: chequea si un acceso dado se permite a un determinado nodo i.

Los includes necesarios son:

```
#include "fs.h"
#include "buf.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "param.h"
#include "super.h"
```

## 6.1.- do_chmod

Realiza la llamada al sistema ***chmod(name, mode)***.

Permite cambiar el modo de acceso para un fichero **name** al superusuario o propietario siempre y cuando no pertenezca a un sistema de archivos de sólo lectura.

- Variables importantes:
  - **rip**, puntero al nodo i del fichero en cuestión.
  - **r**, valor entero que indica el resultado de la operación.

- Algoritmo:

1) Toma el nombre del fichero (*fetch_name*) y lo pone en el *user_path*.

2) Obtener el nodo-i del *user_path (eat_path)* y lo coloca en la variable *rip* del tipo inode.

3) *Si* el uid real es distinto del uid efectivo del proceso *y además* el usuario que va a ejecutar el proceso no es el superusuario *entonces* produce un error.
   *Sino* se comprueba la forma en que se montó el sistema de ficheros que contiene al fichero sobre el que se quiere efectuar el cambio (*read_only*), ya que nadie puede cambiar el modo de un fichero de un FS de solo lectura.

4) *Si* hay error *entonces* se devuelve el i-node a la tabla de i-nodes.

5) Actualiza el modo del i-node haciendo una máscara entre el modo que tiene el fichero y el modo que se le quiere poner (pasado por parámetro).

6) *Si* no es el superusuario y el identificador de grupo real no es el efectivo, es decir, que el proceso que lo abrió no es su propietario, *entonces* se desactiva el bit de grupo utilizando una máscara, actualizando el modo del i-node.

*7)* Actualiza los campos del i-node i_update e i_dirty, para cuando se devuelva el i-node a su tabla, comprobar que si ha cambiado para actulizarlo.

8) Devolver el i-node a la tabla de i-nodes (*put_inode*).

```
/*=================================================================== *
                              do_chmod
*====================================================================*/

PUBLIC int do_chmod()
{
  register struct inode *rip;
  register int r;

  /* Toma nombre fichero */
  if (fetch_name(name, name_length, M3) != OK) return(err_code);

  /* Toma el nodo-i */
  if((rip = eat_path(user_path)) == NIL_INODE) return(err_code);

  /* Si el uid real y efectivo son diferentes y no es el superusuario
  error*/
  if (rip->i_uid != fp->fp_effuid && !super_user)
      r = EPERM;
  else
  /* Comprueba el tipo de sistema de archivos montado */
      r = read_only(rip);

  /* Si hay error retorna el nodo-i */
  if (r != OK)
  {
      put_inode(rip);
      return(r);
  }

  /* Actualiza el modo solicitado */
  rip->i_mode = (rip->i_mode & ~ALL_MODES) | (mode & ALL_MODES);

  /* Se comprueba si se puede cambiar el modo del grupo, solo si es
   el superusuario o si el identificador real es igual al identificador
 efectivo, en otro caso se desactiva el bit de grupo */
  if (!super_user && rip->i_gid != fp->fp_effgid)
             rip->i_mode &= ~I_SET_GID_BIT;

  /* Actualiza campos del i-node */
  rip->i_update |=CTIME;  /* Se actualiza la fecha de modificacion */
  rip->i_dirt = DIRTY;

  /* Devuelve el i-node a la tabla de i-nodes */
  put_inode(rip);

  return(OK);
}
```

## 6.2.- do_chown

Realiza la llamada al sistema *chown (name, owner, group)*.

Permite cambiar el propietario (owner) o grupo (group) para un fichero name, a no ser que el fichero pertenezca a un sistema de archivos de sólo lectura. Si el que quiere hacer el cambio es el superusuario se podrán hacer ambos cambios (propietario y grupo) mientras que si es un usuario normal sólo puede cambiar el grupo de sus propios ficheros.

- Variables importantes:

  - **rip**, puntero al nodo i del fichero en cuestión.
  - **r**, valor entero que indica el resultado de la operación.

- Algoritmo:

1) Toma el nombre del fichero (*fetch_name*) y lo pone en el *user_path*.

2) Obtener el nodo-i del *user_path (eat_path)* y lo coloca en la variable *rip* del tipo inode.

*3)* Se comprueba la forma en que se montó el sistema de ficheros que contiene al fichero sobre el que se quiere efectuar el cambio (*read_only*), ya que nadie puede cambiar un fichero de un FS de solo lectura.

*4)* *Si* el sistema de ficheros está montado como lectura-escritura se continua la ejecución y sino se devuelve el i-node a la tabla de i-nodes (put_inode).

*5)* *Si* es el superusuario *entonces* el identificador de usuario es el propietario que se pasó por parámetro a la función (*owner*).

   *Sino* se comprueba por un lado que el uid real y efectivo del fichero sean iguales, y por otro lado que el uid real del fichero y el owner pasado por parámetro no sean distintos, y por último, que el gid efectivo del fichero y el group pasado por parámetro también sean iguales, ya que sólo los usuarios normales sólo pueden cambiar el grupo de sus propios ficheros.

   Por ejemplo, un usuario normal puede haberse cambiado de grupo, pero un fichero suyo (y que sigue siendo suyo), tiene aún como grupo un grupo equivocado. El usuario puede cambiar el grupo del fichero, siempre que el nuevo grupo del fichero sea en el que él está.

*6)* *Si* no ha habido ningún error (es decir, que el superusuario o el usuario puede cambiar los parámetros) *entonces* se actualizan los campos del i-node: i_gid, i_mode, i_update e i_dirty, para cuando se devuelva el i-node a su tabla, comprobar que si ha cambiado para actulizarlo.

7) Devolver el i-node a la tabla de i-nodes (*put_inode*).

```
/*===========================================================*
 *                      do_chown                             *
 *===========================================================*/
PUBLIC int do_chown()
{
  register struct inode *rip;
  register int r;

  /* Toma el nombre del fichero */
  if(fetch_name(name1, name1_length, M1)!= OK) return(err_code);

  /* Toma el nodo-i */
  if((rip = eat_path(user_path)) == NIL_INODE) return(err_code);

  /* Comprueba el tipo de sistema de archivos montado */
  r = read_only(rip);

  /* Si es un FS montado como lectura-escritura */
  if (r == OK) {

      /* Si es el superusuario, puede hacer todo */
      if (super_user) {

            /* Cambia el propietario */
            rip->i_uid = owner;
      } else {
            /* Los usuarios normales sólo pueden cambiar el
            grupo de sus propios ficheros */
            if (rip->i_uid != fp->fp_effuid) r = EPERM;
            if (rip->i_uid != owner) r = EPERM;
            if (fp->fp_effgid != group) r = EPERM;
      }
  }

  /* Si no ha habido error actualiza campos del i-node */
  if (r == OK) {
      rip->i_gid = group;
      rip->i_mode &= ~(I_SET_UID_BIT | I_SET_GID_BIT);
      rip->i_update |= CTIME;
      rip->i_dirt = DIRTY;
  }

  /* Devuelve el i-node a la tabla de i-nodes */
  put_inode(rip);

  return(r);
}
```

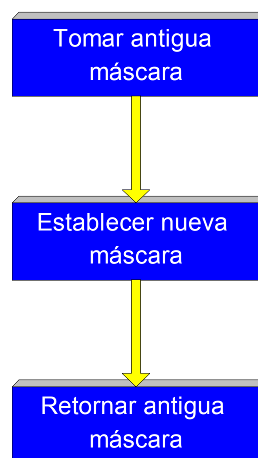
### 6.3.- do_mask

Realiza la llamada al sistema *oldmask=umask (co_mode)*.

Permite cambiar la máscara de limitación de acceso a los ficheros creados por el proceso que le llama (**fp_umask**). La máscara será almacenada en la tabla de procesos, por tanto esta máscara será utilizada para los permisos de los ficheros que cree ese proceso posteriormente. La implementación completa podría hacerse en una sola línea, **fp->fp_umask = ~(co_mode & RWX_MODES)**, pero la llamada debe retornar el valor de la máscara antigua como resultado, por lo que se añaden dos líneas más.

- Variables importantes:
  - **r**, contiene el complementario de la máscara antes de actualizar y será el valor que retorne esta llamada.

- Algoritmo:
  1) Coloca en r el complementario de la máscara de protección de los ficheros creados por el proceso.
  2) Realiza un and entre los permisos solicitados (*co_mode*) con la máscara almacenada *RWX_MODES*, produciendo como resultado la nueva máscara.
  3) Retorna la máscara antigua almacenada en r.



```
/*===========================================================*
 *                       do_umask                                  *
 *===========================================================*/
PUBLIC int do_umask()
{
  register mode_t r;

  /*  Complementa la máscara */
  r = ~fp->fp_umask;

  /* Produce la nueva máscara*/
  fp->fp_umask = ~(co_mode & RWX_MODES);

  /* Retorna la antigua máscara */
  return(r);
}
```

## 6.4.- do_access

Realiza la llamada al sistema *access (name, mode)*.

Permite comprobar si un proceso puede acceder a un archivo en un modo determinado **mode** a un fichero **name.** Esto se comprueba con los identificadores reales del proceso, no con los efectivos.

- Variables importantes:
  - **rip,** puntero al nodo i del fichero en cuestión.
  - **r**, valor entero que indica el resultado de la operación.

- Algoritmo:

  1) Se comprueba si el modo de acceso es correcto (por seguridad), si no se retorna un error.

  2) Toma el nombre del fichero (*fetch_name*) y lo pone en el *user_path*.

  3) Obtener el nodo-i del *user_path (eat_path)* y lo coloca en la variable *rip* del tipo inode.

  4) Chequea los permisos (*forbidden*), comprobando si el modo está permitido mirando su nodo-i.

  5) Devolver el i-node a la tabla de i-nodes (*put_inode*).

  6) Retorna el resultado de forbbiden que indica si modo está permitido o no.

```
/*===========================================================*
 *                      do_access                                 *
 *===========================================================*/
PUBLIC int do_access()
{

  struct inode *rip;
  register int r;

  /* Primero se comprueba si el modo es correcto */
  if ( (mode & ~(R_OK | W_OK | X_OK)) != 0 && mode != F_OK)
      return (EINVAL);

  /* Obtenemos el i-node del fichero cuyo acceso se va a
   chequear, si es posible. */
  if(fetch_name(name, name_length, M3) != OK) return(err_code);
  if((rip = eat_path(user_path)) == NIL_INODE) return(err_code);

  /* Chequea los permisos. */
  r = forbidden(rip, (mode_t) mode);

  put_inode(rip);
  /* Retorna si el modo está permitido o no */
  return(r);
}
```

## 6.5.- Forbidden:

Es llamada por la función anterior (*do_access*).

Permite comprobar si el acceso está prohibido o no, utilizando el modo pasado por parámetro (*access_desired*)**,** para el identificador real o efectivo.

Si la llamada a **forbidden** ha sido efectuada por la función **do_access** se comprueba con el identificador real del proceso, y si no con el efectivo.

- Variables importantes:
  - **rip,** puntero al nodo i del fichero en cuestión.
  - **r**, valor entero que indica el resultado de la operación.
  - **access_desired**, modo de acceso deseado, RWX.
  - **sp**, estructura que va a apuntar al superbloque.

- Algoritmo:

1) Comprobar que el i-node pertenece a un FS montado.

2) *Si* es un sistema montado *entonces*

   Recorrer los superbloques de los FS que están montados:

   *Si* el i-node es el raiz de algún FS montado *entonces* se obtiene el i-node del dispositivo en el que reside, en otro caso sigue con el mismo i-node que se pasó como parámetro (*rip*).

3) Almacena en la variable local *bits* el modo del i-node.

4) Si ha llamado el *do_access* se almacenan en las variables *test_uid* y *test_gid* el uid y gid reales, y si no los efectivos.

5) *Si* el uid anterior es el del superusuario *entonces*
   comprueba que tipo de permisos le puede conceder.
   *Sino*
   Comprueba el tipo de permisos a conceder.

6) *Si* los permisos permitidos no se corresponden con los deseados *entonces* se retorna un error.

7) *Si* el modo de acceso es correcto *entonces*
   *Si* se desea escribir y el bit de escritura está activado *entonces*
   Se comprueba que el FS esté montado como lectura-escritura.

8) Si el i-node pertenecía a un sistema montado (*rip!=old_rip*) *entonces* se devuelve a la tabla de i-nodes

9) Retorna si se permite o no acceder con el modo deseado al fichero.

```
/*===========================================================*
 *                       forbidden                            *
 *===========================================================*/
PUBLIC int forbidden(rip, access_desired)
register struct inode *rip;
mode_t access_desired;
{
  /* La rutina mira el uid del llamador en la tabla fproc. Si el acceso se
permite se retorna OK, en caso contrario se retorna error */

  register struct inode *old_rip = rip;
  register struct super_block *sp;
  register mode_t bits, perm_bits;
  int r, shift, test_uid, test_gid;

  /* Si el i-node pertenece a un sistema de ficheros que está montado. */
  if (rip->i_mount == I_MOUNT)

  /*Recorre los superbloques de los sistemas de ficheros que están
   montados, y comprueba si el i-node rip es el raíz de algún sistema de
   ficheros montado. Si es así, lo obtiene del dispositivo en el que
   reside,y si no es así, sigue con el mismo rip que se pasó como parámetro */
        for (sp = &super_block[1]; sp < &super_block[NR_SUPERS]; sp++)
          if (sp->s_imount == rip) {
                 rip = get_inode(sp->s_dev, ROOT_INODE);
              break;
          }

  /* Toma los bits del modo */
  bits = rip->i_mode;

/* Si quien ha llamado al forbidden es el do_access se
 * comprueba con uid y gid reales del proceso, y si no,
 * se comprueba con los efectivos */
 test_uid=(fs_call == ACCESS ? fp->fp_realuid : fp->fp_effuid);
 test_gid=(fs_call == ACCESS ? fp->fp_realgid : fp->fp_effgid);
```

```
  /* Si es el superusario */
  if (test_uid == SU_UID) {

  /* Concede permiso de lectura/escritura. Concede permisos de búsqueda
  para directorios. Concede permisos de ejecución si y sólo sí uno de
  los bits x está activado(si el fichero tiene algún permiso de
  ejecución */

     if((bits & I_TYPE)==I_DIRECTORY || bits & ((X_BIT << 6)
                       |(X_BIT << 3) | X_BIT))

           perm_bits = R_BIT | W_BIT | X_BIT;
     else
           perm_bits = R_BIT | W_BIT;

 } else {
     if (test_uid == rip->i_uid) shift = 6;            /* propietario */
     else if (test_gid == rip->i_gid ) shift = 3;      /* grupo */
     else shift = 0;                                   /* otro */
     perm_bits = (bits >> shift) & (R_BIT | W_BIT | X_BIT);
  }

  r = OK;
  /* Si el modo permitido es semejante al deseado se continua
   y sino da un error */
  if ((perm_bits | access_desired) != perm_bits) r = EACCES;

  /* Comprueba si alguno está intentando escribir en un sistema
   de ficheros montado como de sólo lectura. */
  if (r == OK)
             if (access_desired & W_BIT) r = read_only(rip);

  /* Si el i-node pertenecía a un FS montado se devuelve a la tabla de
    i-nodes */
  if (rip != old_rip) put_inode(rip);

  /* Se retorna si se permite o no acceder con el modo deseado */
  return(r);
}
```

## 6.6.- Read_only:

Verifica si el fichero está o no montado sobre un sistema de archivos de sólo lectura.

- Variables importantes:
  - **ip**, puntero al i-node cuyo sistema de ficheros se va a chequear.
  - **sp**, puntero al superbloque del dispositivo del i-node.

- Algoritmo:
  1) Obtener superbloque.
  2) Comprobar si el FS sobre el que reside el nodo-i está montado como de lectura o lectura-escritura.

```
/*===========================================================*
 *                      read_only                            *
 *===========================================================*/
PUBLIC int read_only(ip)
struct inode *ip;
{

  register struct super_block *sp;

  /* Obtiene el superbloque */
  sp = ip->i_sp;

  /* Chequea si el FS sobre el que reside el nodo-i está montado como de
   sólo lectura */
  return(sp->s_rd_only ? EROFS : OK);
}
```

# 7.- PREGUNTAS:

### 7.1.- Hacer una traza desde que un usuario hace un chmod hasta que se ejecuta la función do_chmod.

Mirar esquema página 7.

### 7.2.- ¿Qué hace y cómo funciona la función do_chmod?

Mirar apartado 6.1

### 7.3.- ¿Qué hace y cómo funciona la función do_chown?

Mirar apartado 6.2

### 7.4.- ¿Qué hace y cómo funciona la función do_umask?

Mirar apartado 6.3

### 7.5.- ¿Qué hace y cómo funciona la función do_access?

Mirar apartado 6.4

### 7.6.- ¿Qué hace y cómo funciona la función forbidden?

Mirar apartado 6.5