

MÓDULO II. Uso de los Servicios del SO mediante la API

Sesión 2. Llamadas al sistema para el Sistema de Archivos (Parte II)

19/07/2025

Índice

1. Objetivos principales	1
2. Llamadas al sistema relacionadas con los permisos de los archivos	1
2.1 La llamada al sistema umask	1
2.2 Las llamadas al sistema chmod y fchmod	2
📖 Actividad 2.1. Trabajo con llamadas de cambio de permisos	3
3. Funciones de manejo de directorios	5
📖 Actividad 2.2. Trabajo con funciones estándar de manejo de directorios	7

1. Objetivos principales

Esta sesión está pensada para trabajar con las llamadas al sistema que modifican los permisos de un archivo y con los directorios.

- Conocer y saber usar las órdenes para poder modificar y controlar los permisos de los archivos que crea un proceso basándose en la máscara que tiene asociado el proceso.
- Conocer las funciones y estructuras de datos que nos permiten trabajar con los directorios.
- Comprender los conceptos e implementaciones que utiliza un sistema operativo UNIX para construir las abstracciones de archivos y directorios.

2. Llamadas al sistema relacionadas con los permisos de los archivos

En este punto trabajaremos ampliando la información que ya hemos visto en la sesión 1. Vamos a entender por qué cuando un proceso crea un archivo se le asignan a dicho archivo unos permisos concretos. También nos interesa controlar los permisos que queremos que tenga un archivo cuando se crea.

2.1 La llamada al sistema umask

La llamada al sistema umask fija la máscara de creación de permisos para el proceso y devuelve el valor previamente establecido. El argumento de la llamada puede formarse mediante una combinación OR de las nueve constantes de permisos (rwx para ugo) vistas anteriormente. A continuación se muestra un resumen de la página del manual del programador de Linux para esta llamada:

NOMBRE

umask – establece la máscara de creación de ficheros

SINOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask(mode_t mask);
```

DESCRIPCIÓN

umask establece la máscara de usuario a mask & 0777.
La máscara de usuario es usada por open(2) para establecer los permisos iniciales del archivo que se va a crear. Específicamente, los permisos presentes en la máscara se desactivan del argumento mode de open (así pues, por ejemplo, si creamos un archivo con campo mode= 0666 y tenemos el valor común por defecto de umask=022, este archivo se creará con permisos: 0666 & ~022 = 0644 = rw-r--r--, que es el caso más normal).

VALOR DEVUELTO

Esta llamada al sistema siempre tiene éxito y devuelve el valor anterior de la máscara.

2.2 Las llamadas al sistema chmod y fchmod

Estas dos funciones nos permiten cambiar los permisos de acceso para un archivo que existe en el sistema de archivos. La llamada chmod opera sobre un archivo especificado por su *pathname*, mientras que la función fchmod lo hace sobre un archivo que ha sido previamente abierto con open.

A continuación se muestra un resumen de la página del manual del programador de Linux para esta llamada:

NOMBRE

chmod, fchmod – cambia los permisos de un archivo

SINOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

DESCRIPCIÓN

Cambia los permisos del archivo dado mediante path o referido por fildes. Los permisos se pueden especificar mediante un OR lógico de los siguientes valores:

Constante	Octal	Descripción
S_ISUID	04000	Activa la asignación del UID del propietario al UID efectivo del proceso que ejecute el archivo.
S_ISGID	02000	Activa la asignación del GID del propietario al GID efectivo del proceso que ejecute el archivo.

Constante	Octal	Descripción
S_ISVTX	01000	Activa el <i>sticky bit</i> . En directorios, significa que un proceso no privilegiado no puede borrar o renombrar archivos a menos que sea el propietario y tenga permiso de escritura. Por ejemplo, se utiliza en el directorio /tmp.
S_IRWXU	00700	El usuario (propietario del archivo) tiene permisos de lectura, escritura y ejecución.
S_IRUSR	00400	Lectura para el propietario (= S_IREAD no POSIX).
S_IWUSR	00200	Escritura para el propietario (= S_IWRITE no POSIX).
S_IXUSR	00100	Ejecución/búsqueda para el propietario (=S_IEXEC no POSIX).
S_IRWXG	00070	El grupo tiene permisos de lectura, escritura y ejecución.
S_IRGRP	00040	Lectura para el grupo.
S_IWGRP	00020	Escritura para el grupo.
S_IXGRP	00010	Ejecución/búsqueda para el grupo.
S_IRWXO	00007	Otros tienen permisos de lectura, escritura y ejecución.
S_IROTH	00004	Lectura para otros.
S_IWOTH	00002	Escritura para otros.
S_IXOTH	00001	Ejecución/búsqueda para otros.

VALOR DEVUELTO

En caso de éxito, devuelve 0. En caso de error, -1 y se asigna a la variable `errno` un valor adecuado.

Avanzado Para cambiar los bits de permisos de un archivo, el **UID efectivo del proceso** debe ser igual al del propietario del archivo, o el proceso debe tener permisos de *root* o superusuario (**UID efectivo del proceso debe ser 0**).

Si el UID efectivo del proceso no es cero y el grupo del fichero no coincide con el ID de grupo efectivo del proceso o con uno de sus ID's de grupo suplementarios, el bit S_ISGID se desactivará, aunque esto no provocará que se devuelva un error.

Dependiendo del sistema de archivos, los bits S_ISUID y S_ISGID podrían desactivarse si el archivo es escrito. En algunos sistemas de archivos, solo el *root* puede asignar el '*sticky bit*', lo cual puede tener un significado especial (por ejemplo, para directorios, un archivo sólo puede ser borrado por el propietario o el *root*).

Actividad 2.1. Trabajo con llamadas de cambio de permisos

Consulta el manual en línea para las llamadas al sistema `umask` y `chmod`.

Ejercicio 1. ¿Qué hace el siguiente programa?

```
/*
tarea3.c

Trabajo con llamadas al sistema del Sistema de Archivos 'POSIX 2.10 compliant'
```

Este programa fuente está pensado para que se cree primero un programa con la parte de CREACION DE ARCHIVOS y se haga un `ls -l` para fijarnos en los permisos y entender la llamada `umask`.
En segundo lugar (una vez creados los archivos) hay que crear un segundo programa con la parte de CAMBIO DE PERMISOS para comprender el cambio de permisos relativo a los permisos que actualmente tiene un archivo frente a un establecimiento de permisos absoluto.

```
*/  
  
#include <sys/types.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <stdio.h>  
#include <errno.h>  
  
/*  
Definimos los permisos que queremos solicitar (incluyen escritura  
para grupo y otros)  
Permisos Solicitados: 0777 (rwxrwxrwx) - Máximo absoluto para simplificar  
*/  
  
int main(int argc, char *argv[]) {  
    int fd1, fd2;  
    struct stat atributos;  
  
    /*  
    1. Establecer Umask inicial a 0022 (típica) para la prueba de archivo1  
    Esto restringirá la escritura (w) para grupo y otros.  
    */  
    umask(0022);  
  
    printf("--- CREACIÓN DE ARCHIVO1 con umask(0022) ---\n");  
  
    /* CREACION DE ARCHIVO1: Los permisos solicitados serán restringidos */  
    if ((fd1 = open("archivo1", O_CREAT | O_TRUNC | O_WRONLY,  
                    S_IRWXU | S_IRWXG | S_IRWXO)) < 0) {  
        perror("Error en open(archivo1,...)");  
        exit(-1);  
    }  
    close(fd1); /* Cerrar para que los permisos se fijen */  
  
    /* 2. CAMBIAR Umask a 0 (CERO) */  
    printf("--- CAMBIANDO umask a 0 --- \n");  
    umask(0); /* Elimina todas las restricciones de permisos. */  
  
    /* CREACION DE ARCHIVO2: Los permisos solicitados se respetarán completamente */
```

```

if ((fd2 = open("archivo2", O_CREAT | O_TRUNC | O_WRONLY,
               S_IRWXU | S_IRWXG | S_IRWXO)) < 0) {
    perror("Error en open(archivo2,...)");
    exit(-1);
}
close(fd2); /* Cerrar para que los permisos se fijen */

printf("--- RESULTADOS DE LOS PERMISOS DE CREACIÓN ---\n");
/* Mostrar permisos de archivo1 */
if (stat("archivo1", &atributos) == 0) {
    printf("archivo1 (umask 0022): 0%o\n", atributos.st_mode & 0777);
}

/* Mostrar permisos de archivo2 */
if (stat("archivo2", &atributos) == 0) {
    printf("archivo2 (umask 0000): 0%o\n", atributos.st_mode & 0777);
}

/* CAMBIO DE PERMISOS */
if (stat("archivo1", &atributos) < 0) {
    printf("\nError al intentar acceder a los atributos de archivo1");
    perror("\nError en lstat");
    exit(-1);
}

if (chmod("archivo1", (atributos.st_mode & ~S_IXGRP) | S_ISGID) < 0) {
    perror("\nError en chmod para archivo1");
    exit(-1);
}

if (chmod("archivo2", S_IRWXU | S_IRGRP | S_IWGRP | S_IROTH) < 0) {
    perror("\nError en chmod para archivo2");
    exit(-1);
}

return 0;
}

```

3. Funciones de manejo de directorios

Aunque los directorios se pueden leer utilizando las mismas llamadas al sistema que para los archivos normales, como la estructura de los directorios puede cambiar de un sistema a otro, los programas en este caso no serían portables. Para solucionar este problema, se va a utilizar una biblioteca estándar de funciones de manejo de directorios que se presentan de forma resumida a continuación:

- `opendir`: se le pasa el *pathname* del directorio a abrir, y devuelve un puntero a la estructura de tipo `DIR`, llamada *stream* de directorio. El tipo `DIR` está definido en `<dirent.h>`.
- `readdir`: lee la entrada donde esté situado el puntero de lectura de un directorio ya abierto cuyo *stream* se pasa a la función. Después de la lectura adelanta el puntero una posición. Devuelve la entrada leída a

través de un puntero a una estructura (`struct dirent`), o devuelve `NULL` si llega al final del directorio o se produce un error.

- `closedir`: cierra un directorio, devolviendo `0` si tiene éxito, en caso contrario devuelve `-1`.
- `seekdir`: permite situar el puntero de lectura de un directorio (se tiene que usar en combinación con `telldir`).
- `telldir`: devuelve la posición del puntero de lectura de un directorio.
- `rewinddir`: posiciona el puntero de lectura al principio del directorio.

A continuación se dan las declaraciones de estas funciones y de las estructuras que se utilizan, contenidas en los archivos `<sys/types.h>` y `<dirent.h>` y del tipo `DIR` y la estructura `dirent` (entrada de directorio).

```
DIR *opendir(char *dirname)
struct dirent *readdir(DIR *dirp)
int closedir(DIR *dirp)
void seekdir(DIR *dirp, long loc)
long telldir(DIR *dirp)
void rewinddir(DIR *dirp)
```

```
typedef struct _dirdesc {
    int dd_fd;
    long dd_loc;
    long dd_size;
    long dd_bbase;
    long dd_entno;
    long dd_bsize;
    char *dd_buf;
} DIR;
```

/ La estructura struct dirent conforme a POSIX 2.1 es la siguiente: */*

```
#include <sys/types.h>
#include <dirent.h>
struct dirent {
    long d_ino; /* número i-nodo */
    char d_name[256]; /* nombre del archivo */
};
```

Veamos un ejemplo de programa que recorre un directorio:

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>

int main(int argc, char *argv[]) {
    DIR *dir_ptr;
    struct dirent *entry;

    if (argc != 2) { /* Comprobar el número de argumentos */
        fprintf(stderr, "Uso: %s <ruta_directorio>\n", argv[0]);
```

```

    return 1;
}

const char *directorio = argv[1];

dir_ptr = opendir(directorio); /* Abrir directorio */

if (dir_ptr == NULL) {
    perror("Error al abrir el directorio");
    return 1;
}

printf("Contenido del directorio '%s':\n", directorio);
printf("-----\n");

while ((entry = readdir(dir_ptr)) != NULL) { /* Recorrer el directorio */
    printf("%s\n", entry->d_name);
}

if (closedir(dir_ptr) == -1) {
    perror("Error al cerrar el directorio");
    return 1;
}

return 0;
}


```

El programa intentará abrir el directorio especificado por argumento `argv[1]` usando la función `opendir()`. Si devuelve `NULL` (porque el directorio no existe, no tienes permisos, o no es un directorio), imprime un mensaje de error del sistema (usando `perror`) y termina. En el bucle, leemos la siguiente “entrada” (archivo, subdirectorio, enlace, etc.) dentro del directorio abierto. El bucle se repite mientras `readdir` siga encontrando entradas. Cuando no hay más entradas que leer, `readdir` devuelve `NULL` y el bucle termina.

Una vez terminado el bucle, debemos cerrar el puntero al directorio usando `closedir()` para liberar los recursos, y en cualquier caso, es una buena praxis.

Actividad 2.2. Trabajo con funciones estándar de manejo de directorios

Mirad las funciones estándar de trabajo con directorios utilizando `man` `opendir` y viendo el resto de funciones que aparecen en la sección **VÉASE TAMBIÉN** de esta página del manual.

 **Ejercicio 2.** Realiza un programa en C utilizando las llamadas al sistema necesarias que acepte como entrada:

- Un argumento que representa el ‘**pathname**’ de un directorio.
- Otro argumento que es un **número octal de 4 dígitos** (similar al que se puede utilizar para cambiar los permisos en la llamada al sistema `chmod`). Para convertir este argumento tipo cadena a un tipo numérico puedes utilizar la función `strtol`. **Consulta el manual en línea para conocer sus argumentos.**

El programa tiene que usar el número octal indicado en el segundo argumento para cambiar los permisos de todos los archivos que se encuentren en el directorio indicado en el primer argumento.

El programa debe proporcionar en la salida estándar una línea para cada archivo del directorio que esté formada por:

`<nombre_de_archivo> : <permisos_antiguos> <permisos_nuevos>`

Si no se pueden cambiar los permisos de un determinado archivo se debe especificar la siguiente información en la línea de salida:

`<nombre_de_archivo> : <errno> <permisos_antiguos>`



Ejercicio 3. Programa una nueva orden que recorra la jerarquía de subdirectorios existentes a partir de uno dado como argumento y devuelva la cuenta de todos aquellos archivos regulares que tengan permiso de ejecución para el grupo y para otros. Además del nombre de los archivos encontrados, deberá devolver sus números de inodo y la suma total de espacio ocupado por dichos archivos. El formato de la nueva orden será:

`./buscar <pathname>`

donde `<pathname>` especifica la ruta del directorio a partir del cual queremos que empiece a analizar la estructura del árbol de subdirectorios. En caso de que no se le de argumento, tomará como punto de partida el directorio actual. Ejemplo de la salida después de ejecutar el programa:

Los i-nodos son:

```
./a.out 55
./bin/ej 123
./bin/ej2 87
...
```

Existen 24 archivos regulares con permiso x para grupo y otros
El tamaño total ocupado por dichos archivos es 2345674 bytes