Modulo-II-Sesion-2.pdf

KIKONASO

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

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

Sesión 1. Llamadas al sistema para el SA (Parte II)

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>

int main(int argc, char *argv[])
{
int fd1,fd2;
struct stat atributos;

//CREACION DE ARCHIVOS
if( (fd1=open("archivo1",O_CREAT|O_TRUNC|O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP))<0)
{
      printf("\nError %d en open(archivo1,...)",errno);
      perror("\nError en open");
      exit(-1);
}
umask(0);
if( (fd2=open("archivo2",O_CREAT|O_TRUNC|O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP))<0)
{
      printf("\nError %d en open(archivo2,...)",errno);
      perror("\nError en open");
      exit(-1);
}

//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);
}
close(fd1);
close(fd2);

return 0;
}
```

El programa se basa en el uso de llamadas al sistema del Sistema de Archivos para crear dos archivos y modificarle los permisos, además de ver cómo funciona la máscara umask:

1. Definición de variables:

- **int fd1, fd2:** Estas son variables de tipo descriptor de archivo que se usarán para crear y abrir los archivos.
- **struct stat atributos:** Esta estructura almacena los atributos de **archivo1**, como los permisos actuales, para realizar cambios relativos a estos.

2. Creación de archivos:

- El programa crea el archivo **archivo1** con permisos específicos de grupo: lectura (**S_IRGRP**), escritura (**S_IWGRP**) y ejecución (**S_IXGRP**). Si hay un error al abrir o crear el archivo, muestra un mensaje de error y termina el programa.
- Luego, se llama a **umask(0);**. Esta función establece la máscara de permisos de creación de archivos a 0, lo que significa que no se restringirán los permisos que se indiquen explícitamente al crear archivos.
- A continuación, se crea **archivo2** con los mismos permisos de grupo que **archivo1**. Debido a que la máscara es 0, se establecerán exactamente los permisos indicados.

3. Cambio de permisos:

- Se usa **stat** para obtener los atributos actuales de **archivo1** y almacenarlos en **atributos**.
- Luego, se llama a chmod para modificar los permisos de **archivo1**. La operación realiza dos cambios:
    - Elimina el permiso de ejecución para el grupo (**~S_IXGRP**).
    - Establece el **bit de setgid** (**S_ISGID**), que indica que los archivos creados en el directorio heredan el grupo propietario.
- Para **archivo2**, se usa **chmod** para establecer permisos absolutos: lectura, escritura y ejecución para el usuario propietario (**S_IRWXU**), lectura y escritura para el grupo (**S_IRGRP | S_IWGRP**), y solo lectura para otros (**S_IROTH**).

4. Cierre de archivos:

- Finalmente, se cierran los descriptores de archivo **fd1** y **fd2**.

Una vez compilado y ejecutado el programa obtenemos la siguiente salida:

./tarea1

ls -l

total 20

----rwS--- 1 user user 0 nov 14 10:47 archivo1

-rwxrw-r-- 1 user user 0 nov 14 10:47 archivo2

-rwxrwxr-x 1 user user 16360 nov 14 10:33 tarea1

-rw-rw-r-- 1 user user 1784 nov 14 10:36 tarea1c.c

Comprobamos que:

archivo1: ----rwS---

- Estos permisos indican:
    - Sin permisos para el usuario (ni lectura, ni escritura, ni ejecución).
    - Lectura y escritura para el grupo.
    - Sin permisos para otros.
    - El bit **S** en el campo de ejecución del grupo significa que el bit de **setgid** está activo, pero no hay permiso de ejecución para el grupo. Este bit **S** indica que el archivo tiene el bit de **setgid** establecido sin ser ejecutable.

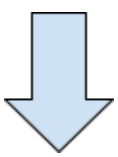
archivo2: -rwxrw-r--

- Los permisos indican:
    - Lectura, escritura y ejecución para el usuario.
    - Lectura y escritura para el grupo.
    - Solo lectura para otros.

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>**

En la página de abajo dejo la implementación del programa



```c
/*
 tarea2.c
 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>
 */

#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

int main(int argc, char *argv[]){

   // Verificación de argumentos
    if (argc!=3){
            perror("\nError, el programa tiene que ser usado con al menos un parámetro. Uso: ./tarea2 <ruta_directorio> <n_octal_permisos> \n");
            exit(-1);
    }

    //Convertir permisos a número octal
        //Uso mode_t porque es el tipo de dato estándar en POSIX para representar permisos de archivos en el sistema.
        //Este tipo es una forma de garantizar que los valores de permisos sean compatibles con las llamadas al sistema
        //que requieren permisos, como chmod.
    mode_t permisos=strtol(argv[2], NULL, 8);

    //Abro el directorio
    DIR *dir= opendir(argv[1]);
    if (dir==NULL){
        perror("Error al abrir el directorio \n");
        exit(-1);
    }

    struct dirent *entrada;
    struct stat metadatos;

    char ruta[1024];  //1024 para que pueda contener toda la ruta de cada archivo, por muy largo que sea

    //Leer cada archivo en el directorio
    while(((entrada=readdir(dir))!=NULL )){
        if(strcmp(entrada->d_name, ".") == 0 || strcmp(entrada->d_name, "..") == 0 ){
            continue;
        }

        // Construir la ruta completa al archivo
        snprintf(ruta, sizeof(ruta), "%s/%s", argv[1], entrada->d_name);

        // Obtener los permisos actuales
        if (stat(ruta, &metadatos) < 0) {
            printf("%s : %d %o\n", entrada->d_name, errno, metadatos.st_mode & 0777);
            continue;
        }

        //Cambiar los permisos
        int resultado= chmod(ruta, permisos );
        if (resultado < 0){
            printf("%s : %d %o\n", entrada->d_name, errno, metadatos.st_mode & 0777);
        } else {
            printf("%s : %o %o\n", entrada->d_name, metadatos.st_mode & 0777, permisos);
        }

    }

    //Cerrar el directorio
    closedir(dir);

    return 0;
}

```

Ahora vemos como funciona:

ls -l

total 44

-----w--w- 1 user user 0 nov 14 10:47 archivo1

-----w--w- 1 user user 0 nov 14 10:47 archivo2

-----w--w- 1 user user 16360 nov 14 10:33 tarea1

-----w--w- 1 user user 1784 nov 14 10:36 tarea1c.c

-rwxrwxrwx 1 user user 16480 nov 14 11:21 tarea2

-----w--w- 1 user user 3156 nov 14 11:21 tarea2.c

user@Lenovo-Usuario:~/Documentos/2ºCARRERA/SO/ModuloII/Sesión 2$ ./tarea2 . 0777

tarea1 : 22 777

tarea2 : 777 777

archivo1 : 22 777

archivo2 : 22 777

tarea2.c : 22 777

tarea1c.c : 22 777

user@Lenovo-Usuario:~/Documentos/2ºCARRERA/SO/ModuloII/Sesión 2$ ls -l

total 44

-rwxrwxrwx 1 user user   0 nov 14 10:47 archivo1

-rwxrwxrwx 1 user user   0 nov 14 10:47 archivo2

-rwxrwxrwx 1 user user 16360 nov 14 10:33 tarea1

-rwxrwxrwx 1 user user 1784 nov 14 10:36 tarea1c.c

-rwxrwxrwx 1 user user 16480 nov 14 11:21 tarea2

-rwxrwxrwx 1 user user 3156 nov 14 11:21 tarea2.c

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>

Dónde <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:

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

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

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

void buscar_archivos(const char *pathname, int *tam, int *contador_f) {
    DIR *dir;
    struct dirent *entrada;
    struct stat metadatos;
    char ruta[1024];

    // Abrir el directorio
    if ((dir = opendir(pathname)) == NULL) {
        perror("Error al abrir el directorio");
        return;
    }

    // Recorrer el directorio
    while ((entrada = readdir(dir)) != NULL) {
        // Evitar "." y ".."
        if (entrada->d_name[0] == '.') continue;

        // Formar la ruta completa del archivo
        snprintf(ruta, sizeof(ruta), "%s/%s", pathname, entrada->d_name);

        // Obtener los atributos del archivo
        if (stat(ruta, &metadatos) == -1) {
            perror("Error al obtener atributos del archivo");
            continue;
        }

        // Comprobar si es un archivo regular
        if (S_ISREG(metadatos.st_mode)) {
            // Verificar los permisos de ejecución para grupo y otros
            if ((metadatos.st_mode & S_IXGRP) && (metadatos.st_mode & S_IXOTH)) {
                // Imprimir el nombre del archivo y el número de inodo
                printf("%s %lu\n", ruta, (unsigned long)metadatos.st_ino);
                // Sumar el tamaño al total y aumentar el contador
                *tam += metadatos.st_size;
                (*contador_f)++;
            }
        }
        // Si es un directorio, hacer una llamada recursiva
        else if (S_ISDIR(metadatos.st_mode)) {
            buscar_archivos(ruta, tam, contador_f);
        }
    }

    // Cerrar el directorio
    closedir(dir);
}

int main(int argc, char *argv[]) {
    const char *dir_origen;
    int tam = 0, contador_f = 0;

    // Si no se da un directorio como argumento, tomar el directorio actual
    if (argc < 2) {
        dir_origen = ".";
    } else {
        dir_origen = argv[1];
    }

    // Llamar a la función que recorre los directorios
    buscar_archivos(dir_origen, &tam, &contador_f);

    // Imprimir el resumen
    printf("\nExisten %d archivos regulares con permiso x para grupo y otros\n", contador_f);
    printf("El tamaño total ocupado por dichos archivos es %d bytes\n", tam);

    return 0;
}

```

Ejemplo de ejecución:

./buscar ..

../Sesión 2/buscar 2937611

../Sesión 2/tarea1 2889093

../Sesión 2/tarea2 2885800

../Sesión 2/archivo1 2937607

../Sesión 2/archivo2 2937608

../Sesión 2/tarea2.c 2937609

../Sesión 2/tarea1c.c 2893609

Existen 7 archivos regulares con permiso x para grupo y otros

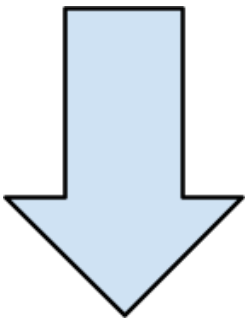
El tamaño total ocupado por dichos archivos es 54076 bytes

Ejercicio 4. Implementa de nuevo el programa buscar del ejercicio 3 utilizando la llamada al sistema nftw.

Voy a proporcionar el código abajo, he tenido problemas a la hora de compilarlo con gcc, prueba a hacerlo de manera normal y si te falla usa el comando:

gcc tarea4.c -o buscar_mejorado -std=gnu99 -D_XOPEN_SOURCE=500

La bandera -D_XOPEN_SOURCE le indica al compilador que incluya las funciones y constantes POSIX necesarias, como **nftw** y **FTW_PHYS**, que de otra forma pueden estar limitadas por defecto.



```c
/*
 tarea4.c
 Implementa de nuevo el programa buscar del ejercicio 3 utilizando la llamada al sistema nftw.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ftw.h>  // Incluir ftw.h para nftw y FTW_PHYS

int tam = 0;            // Variable global para almacenar el tamaño total
int contador_f = 0;     // Variable global para contar archivos que cumplen los permisos

// Función que nftw llama para cada archivo o directorio encontrado
int visitar(const char* path, const struct stat* stat, int flags, struct FTW* ftwbuf) {
    // Comprobar si es un archivo regular
    if (S_ISREG(stat->st_mode)) {
        // Verificar permisos de ejecución para grupo y otros
        if ((stat->st_mode & S_IXGRP) && (stat->st_mode & S_IXOTH)) {
            // Imprimir la ruta y el número de inodo
            printf("%s %lu\n", path, (unsigned long)stat->st_ino);
            // Sumar el tamaño del archivo al total y aumentar el contador de archivos
            tam += stat->st_size;
            contador_f++;
        }
    }
    return 0;  // Continuar el recorrido
}

int main(int argc, char** argv) {
    // Directorio de origen: si no se especifica, se usa el directorio actual
    const char *dir_origen = (argc >= 2) ? argv[1] : ".";

    // Llamada a nftw para recorrer el árbol de directorios
    if (nftw(dir_origen, visitar, 10, FTW_PHYS) != 0) {
        perror("Error en nftw");
        return 1;
    }

    // Imprimir el resumen de archivos y tamaño total
    printf("\nExisten %d archivos regulares con permiso x para grupo y otros\n", contador_f);
    printf("El tamaño total ocupado por dichos archivos es %d bytes\n", tam);

    return 0;
}

```