Ejercicios Prácticas SO

Grado Ingeniería Informática

E.T.S. Ing. Informática y de Telecomunicación (ETSIIT)

Granada

Alejandro Alcalde
elbauldelprogramador.com

Ejercicios sesiones Prácticas

Alejandro Alcalde

[1]ETSIIT, Granada.
E-mail address: algui91@gmail.com.

November 23, 2013

LECTURE 2

Sesión 1

Exercise 1. ¿Qué hace el siguiente programa? Probad tras la ejecución del programa las siguientes órdenes del shell: **$>cat archivo y $> od -c archivo**

```c
/*
tarea1.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
Probar tras la ejecución del programa: >cat archivo y > od -c archivo
*/

#include<sys/types.h>   //Primitive system data types for abstraction of implementation-dependent data types.
                        //POSIX Standard: 2.6 Primitive System Data Types <sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>

char buf1[]="abcdefghij";
char buf2[]="ABCDEFGHIJ";

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

if( (fd=open("archivo",O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR))<0) {
    printf("\nError %d en open",errno);
    perror("\nError en open");
    exit(-1);
}
if(write(fd,buf1,10) != 10) {
    perror("\nError en primer write");
    exit(-1);
}

if(lseek(fd,40,SEEK_SET) < 0) {
    perror("\nError en lseek");
    exit(-1);
}

if(write(fd,buf2,10) != 10) {
    perror("\nError en segundo write");
    exit(-1);
}

return 0;
}
```

El programa crea un archivo llamado **archivo**, escribe el primer buffer, que contiene **abcdefghij**, éstos caracteres se escriben desde la primera posición del archivo, luego, mueve el puntero a la posición 40 y escribe el segundo buffer, que contiene **ABCDEFGHIJ**, por tanto, el fichero va a tener contenido nulo en las posiciones 11-39. Con el comando **cat** no se aprecia el contenido nulo porque es ignodaro, sin embargo con od sí se muestran:

```
 od -c archivo
0000000   a   b   c   d   e   f   g   h   i   j  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0000060   I   J
0000062
```

Exercise 2. Implementa un programa que acepte como argumento un "pathname", abra el archivo correspondiente y utilizando un tamaño de lectura en bloques de 80 Bytes cree un archivo de salida en el que debe aparecer lo siguiente:

```
Bloque 1
//los primeros 80 Bytes
Bloque 2
//los siguientes 80 Bytes
...
Bloque n
//los siguientes 80 Bytes
```

Ejercicio 2.c

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

void print_error(char *where){
    printf("\nError %d en %s",errno, where);
    perror("\nError");
    exit(-1);
}

int main(int argc, char *argv[]) {
    int fd;
    if (argc < 2){
        printf("Uso: %s <nombre_fichero> ", argv[0]);
        exit(-1);
    }

    if( (fd=open(argv[1],O_RDONLY))<0) {
    }

    /* Crear el archivo donde se escribira el contenido */
    uint fd_salida;

    if( (fd_salida = open("dump",O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR))<0)
        print_error("open");

    char myBuff[80];
    unsigned int i = 1;
```

```
35      char bloque[11];
36      int leido;
37
38      char *resumen = "El número de bloques es %d";
39
40      if (lseek(fd_salida, 50, SEEK_SET) < 0 )
41          print_error("lseek");
42
43      while ((leido = read(fd, myBuff, 80)) > 0){
44          sprintf(bloque, "\nBloque %d\n", i++);
45          write(fd_salida, bloque, 11);
46          write(fd_salida, myBuff, leido);
47      }
48
49      if(lseek(fd_salida, 0, SEEK_SET) < 0)
50          print_error("lseek");
51
52      sprintf(bloque, "El número de bloques es %d", i-1);
53      write(fd_salida, bloque, strlen(bloque));
54
55      if(close(fd_salida) < 0)
56          print_error("close");
57      if(close(fd) < 0 )
58          print_error("close");
59
60      return 0;
61  }
```

¿Cómo tendrías que modificar el programa para que una vez finalizada la escritura en el archivo de salida y antes de cerrarlo, pudiésemos indicar en su primera línea el número de etiquetas "Bloque i" escritas de forma que tuviese la siguiente apariencia?:

Implementado en la versión de arriba

Exercise 3. ¿Qué hace el siguiente programa?

```
/*
tarea2.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
*/

#include<sys/types.h>    //Primitive system data types for abstraction of implementation-dependent data types.
                         //POSIX Standard: 2.6 Primitive System Data Types <sys/types.h>
#include<unistd.h>         //POSIX Standard: 2.10 Symbolic Constants         <unistd.h>
#include<sys/stat.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>

int main(int argc, char *argv[])
{
int i;
struct stat atributos;
char tipoArchivo[30];

if(argc<2) {
    printf("\nSintaxis de ejecucion: tarea2 [<nombre_archivo>]+\n\n");
    exit(-1);
}
for(i=1;i<argc;i++) {
    printf("%s: ", argv[i]);
    if(lstat(argv[i],&atributos) < 0) {
```

```
        printf("\nError al intentar acceder a los atributos de %s",argv[i]);
        perror("\nError en lstat");
    }
    else {
        if(S_ISREG(atributos.st_mode)) strcpy(tipoArchivo,"Regular");
        else if(S_ISDIR(atributos.st_mode)) strcpy(tipoArchivo,"Directorio");
        else if(S_ISCHR(atributos.st_mode)) strcpy(tipoArchivo,"Especial de caracteres");
        else if(S_ISBLK(atributos.st_mode)) strcpy(tipoArchivo,"Especial de bloques");
        else if(S_ISFIFO(atributos.st_mode)) strcpy(tipoArchivo,"Tubera con nombre (FIFO)");
        else if(S_ISLNK(atributos.st_mode)) strcpy(tipoArchivo,"Enlace relativo (soft)");
        else if(S_ISSOCK(atributos.st_mode)) strcpy(tipoArchivo,"Socket");
        else strcpy(tipoArchivo,"Tipo de archivo desconocido");
        printf("%s\n",tipoArchivo);
    }
}

return 0;
}
```

Para cada archivo que se le pase como parámetro, muestra qué tipo de ficheros son, en este caso usando la llamada **lstat**, que se diferencia de **stat** en que si el archivo es un enlace simbólico, **lstat** proporciona los atributos del enlace en sí, no del archivo al que referencia, ejemplo de ejecución:

```
./tarea2 *
dump: Regular
ej2: Regular
ej2.c: Regular
ej2.c~: Regular
fichero: Regular
fichero~: Regular
tarea1.c: Regular
tarea2: Regular
tarea2.c: Regular
tarea3.c: Regular
```

Exercise 4. Define una macro en lenguaje C que tenga la misma funcionalidad que la macro S_ISREG(mode) usando para ello los flags definidos en <sys/stat.h> para el campo st_mode de la struct stat, y comprueba que funciona en un programa simple. Consulta en un libro de C o en internet cómo se especifica una macro con argumento en C.

Ejercicio 4.c

```
1  #include<sys/types.h>
2  #include<unistd.h>
3  #include<sys/stat.h>
4  #include<stdio.h>
5  #include<errno.h>
6  #include<string.h>
7
8  #define MY_S_ISREG(mode) ((mode & S_IFMT ) == S_IFREG)
9
10 int main(int argc, char *argv[])
11 {
12 int i;
13 struct stat atributos;
14 char tipoArchivo[30];
15
16 if(argc<2) {
17         printf("\nSintaxis de ejecucion: tarea2 [<nombre_archivo>]+\n\n");
18         exit(-1);
19 }
20 for(i=1;i<argc;i++) {
```

```
21          printf("%s: ", argv[i]);
22          if(lstat(argv[i],&atributos) < 0) {
23                  printf("\nError al intentar acceder a los atributos de %s",argv[i]);
24                  perror("\nError en lstat");
25          }
26          else {
27          printf("Usado macro del sistema \n");
28          if(S_ISREG(atributos.st_mode)) strcpy(tipoArchivo,"Regular");
29          printf("%s\n",tipoArchivo);
30          printf("Usando mi macro\n");
31          if(MY_S_ISREG(atributos.st_mode)) strcpy(tipoArchivo,"My Regular");
32          printf("%s\n",tipoArchivo);
33          }
34  }
35
36  return 0;
37  }
```

LECTURE 3

Sesión 2

Exercise 5. ¿Qué hace el siguiente programa?

```c
/*
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>   //Primitive system data types for abstraction of implementation-dependent data types.
                        //POSIX Standard: 2.6 Primitive System Data Types <sys/types.h>
#include<unistd.h>      //POSIX Standard: 2.10 Symbolic Constants         <unistd.h>
#include<sys/stat.h>
#include<fcntl.h>       //Needed for open
#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);
}

return 0;
}
```

Crea dos archivos, **archivo1** con permisos **O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP**, es decir, **---rwx---**, Este valor se logra con la siguiente operación sobre el **umask**, que en mi máquina por defecto vale **0002**:

```
(000000000100000 | 000000000010000 | 000000000001000)
&           ~000000000000010
= (000000000111000) & 111111111111101
= 000000000111000
```

y luego se cambia el **umask** a **0** y se crea **archivo2** con permisos **S_IRGRP|S_IWGRP|S_IXGRP**, aplicando la máscara 0 se obtiene

```
(000000000100000 | 000000000010000 | 000000000001000)
&           ~000000000000000
= 000000000111000 & 111111111111111
= 000000000111000
```

que es **---rwx---**.

La segunda parte del programa, obtiene los atributos de **archivo1**, y luego le cambia los permisos con **(atributos.st_mode & ~S_IXGRP)**, que quiere decir: Si ya tiene permisos de ejecución para el grupo, se desactivan, en caso de no tenerlo, lo activa. Y **| S_ISGID** activa el grupo efectivo.

Exercise 6. 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 u*n* ***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>
```

Ejercicio2.c

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

```c
#include<dirent.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
#include <limits.h>

#include "utils.h"

int main(int argc, char *argv[])
{
    if(argc < 3) {
        fprintf(stderr, "\nUso: %s <path> <permisos en octal>\n\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *path  = (char*) ec_malloc(strlen(argv[1]));
    strcpy(path, argv[1]);

    long permisos;
    char *str = argv[2];
    char *endptr;

    errno = 0;    /* To distinguish success/failure after call */
    permisos = strtol(str, &endptr, 8);

    if ((errno == ERANGE && (permisos == LONG_MAX || permisos == LONG_MIN))
            || (errno != 0 && permisos == 0)) {
        perror("strtol");
        exit(EXIT_FAILURE);
    }

    if (endptr == str) {
        fprintf(stderr, "No se han encontrado dígitos\n");
        exit(EXIT_FAILURE);
    }

    /* SI hemos llegado hasta aquí, el número para los permisos es correcto */

    DIR *directorio = opendir(path);
    struct dirent *contenido = readdir(directorio);
    struct stat atributos;

    /* Ignoramos . y .. */
    while(contenido->d_type == 0x4)
        contenido = readdir(directorio);

    char* path_relativo = (char*) ec_malloc(strlen(path) + 256); //256 es la longitud maxima del nombre de un

    while (contenido){
        strcpy(path_relativo, path);
        strcat(path_relativo, contenido->d_name);

        if (stat(path_relativo, &atributos) < 0)
            fatal("Obteniendo atributos del archivo");

        long permisos_antiguos = atributos.st_mode & (S_IRWXU | S_IRWXG | S_IRWXO); //0777

        if (chmod(path_relativo, permisos) < 0){
            fprintf(stderr, "Error cambiando permisos de %s con permisos %o, el error es %d :",
                    contenido->d_name, permisos_antiguos, errno);
            perror(path_relativo);
```

```
65          } else
66              printf("%s: %o %o\n", contenido->d_name, permisos_antiguos, permisos);
67
68          contenido = readdir(directorio);
69      }
70
71      free(path);
72      free(path_relativo);
73
74      return 0;
75  }
```