SESIÓN 1 MODULO II SO

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

```
/*
tarea1.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
Probad tras la ejecución del programa: $>cat archivo y $> od -c archivo
*/
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.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);
}
close(fd);
return 0;
}
```

En la terminal ejecuto lo siguiente:

```
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ gcc ejercicio1.c -o ejercicio1
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ ls
ejercicio1  ejercicio1.c
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ ./ejercicio1
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ ls
archivo  ejercicio1  ejercicio1.c
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ cat archivo
abcdefghijABCDEFGHIJmorris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$ 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   A   B   C   D   E   F   G   H
0000060   I   J
0000062
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion1$
```

Este programa lo que hace es:

- Crea 2 arrays buf1 y buf2, cada uno de ellos contiene 10 caracteres.

- Crea un entero fd, al cual le asigna la llamada al sistema open, cuyo archivo es “archivo” y si es menor que 0 no se crea. Los flags que se le pasan son O_CREAT, se crea si no existe; O_TRUNC, si existe el fichero y tiene habilitada la escritura, la sobreescribe a tamaño 0; O_WRONLY, solo se permite escritura; S_IRUSR, se

comprueba que el usuario tiene permiso de lectura; S_IWUSR, se comprueba que el usuario tiene permiso de escritura.

- Comprueba que si tras hacer el write del buf1, este tiene un resultado distinto de 10, se mostrará un error, puesto que hemos escrito los 10 caracteres del buf1 en el primer archivo.

- Con lseek, ponemos el puntero del archivo en la posición 40 desde SEEK_SET(inicio del fichero), y daría error si el resultado de la operación es menor que 0.

- Por último escribe en fd el buf2, desde la posición que hemos apuntado con lseek, los 10 caracteres del buf2.

Si compilamos y ejecutamos, se crea el archivo “archivo” y se comprueba que está bien impreso utilizando tanto cat como od -c, tal como se puede ver arriba.

Ejercicio 2. Implementa un programa que realice la siguiente funcionalidad.
El programa acepta como argumento el nombre de un archivo (pathname), lo abre y lo lee en bloques de tamaño 80 Bytes, y crea un nuevo archivo de salida, salida.txt, en el que debe aparecer la siguiente información:
Bloque 1
// Aquí van los primeros 80 Bytes del archivo pasado como argumento.

Bloque 2
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.

Bloque n
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento.

Si no se pasa un argumento al programa se debe utilizar la entrada estándar como archivo de entrada.

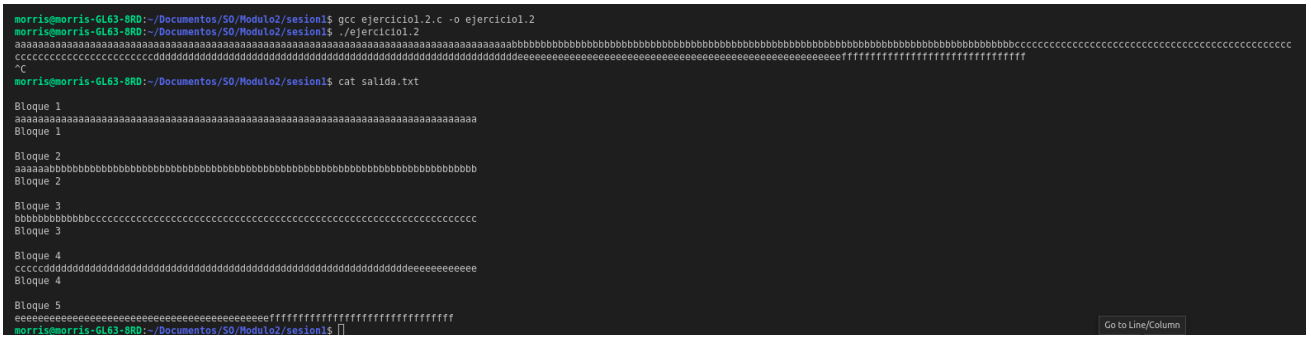
Modificación adicional. ¿Cómo tendrías que modificar el programa para que una vez finalizada la escritura en el archivo de salida y antes de cerrarlo, pudiésemos indicar en su primera línea el número de etiquetas "Bloque i" escritas de forma que tuviese la siguiente apariencia?:
El número de bloques es
Bloque 1
// Aquí van los primeros 80 Bytes del archivo pasado como argumento.

Bloque 2
// Aquí van los siguientes 80 Bytes del archivo pasado como argumento. …

La implementación del ejercicio sería la siguiente:

```
> Modulo2 > sesion1 > C ejercicio1.2.c > ...
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h> //para strlen

//duda: si no paso argumentos no me coge la entrada estándar.

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

    int filein = STDIN_FILENO;
    int fileout;
    int leer;
    int count = 80;
    char buff[80];
    char bloque[15];
    int num_bloque = 0;

    if (argc > 2){
        printf("Error: numero incorrecto de argumentos\n");
        exit(EXIT_FAILURE);
    }

    else if(argc==2){
        if ((filein = open(argv[1], O_RDONLY))< 0){
            printf("Error %d en open\n", errno);
            perror("Error en open\n");
            exit(EXIT_FAILURE);
        }
    }

    if ((fileout = open("salida.txt", O_CREAT|O_TRUNC|O_WRONLY, S_IRUSR|S_IWUSR)) < 0){
        printf("Error %d en open\n", errno);
        perror("Error en open\n");
        exit(EXIT_FAILURE);
    }

    while ((leer = read(filein, buff, count)) != 0){
        if (leer < 0){
            perror("\nError en el read");
            exit(EXIT_FAILURE);
        }

        num_bloque++;

        if (write(fileout, bloque, strlen(bloque)) != strlen(bloque)){
            perror("\nError en write1");
            exit(EXIT_FAILURE);
        }

        sprintf(bloque, "\nBloque %d\n", num_bloque);


        if (write(fileout, bloque, strlen(bloque)) != strlen(bloque)){
            perror("\nError en write2");
            exit(EXIT_FAILURE);
        }


        if (write(fileout, buff, leer) != leer){
            perror("\nError en write3");
             exit(EXIT_FAILURE);
        }

    }

    close(filein);
    close(fileout);
    return EXIT_SUCCESS;
}
```

Si le pasamos argumentos, el resultado quedaría de la siguiente forma:



Explicación del código:

- Si el número de argumentos (argc) que se le pasa es mayor que 2, el número de argumentos es incorrecto, puesto que argv[0] sería el nombre del programa, y argv[1] sería el argumento que se le pasa (en este caso o nada, que tomaría la entrada estándar, o un bloque de letras que dividiría luego en varios bloques).

- Si el número de argumentos es 2, filein toma el argumento dado argv[1], y O_RDONLY, solo se establece permiso de lectura. Si esto es menor que 0 (negativo, que es lo contrario a lo que debería de devolver la función open), habrá un error.

- Hacemos un if de la salida que vamos a crear, con fileout y un open. Si el archivo salida.txt, se crea si no existe (O_CREAT), si existe el fichero y tiene habilitada la escritura, la sobreescribe a tamaño 0 O_TRUNC, O_WRONLY, solo se establece permiso de escritura, y S_IRUSR S_IWUSR, se comprueba que el usuario tiene permiso de lectura y de escritura. Si todo eso es menor que 0 se producirá un error.

- Procedemos a leer el filein que hemos creado antes con argv[1], o sea, procedemos a leer el gran bloque de caracteres. Lee los “count” (80) bytes del filein, o sea lee 80 bytes de los bloques de caracteres, en el buffer buff, mientras todo eso sea distinto de 0:

    - si leer es < 0 → error, ya que no puede dar un numero negativo
    - num_bloque++, empieza por el bloque 1.
    - escribe con write en el “fileout”

Ejercicio 3. ¿Qué hace el siguiente programa?

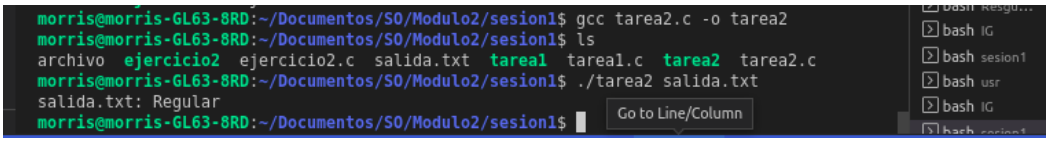
```
tarea2.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
*/
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
int main(int argc, char *argv[])
{
int i;
struct stat atributos;
char 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,"Cauce con nombre
(FIFO)");
     else if(S_ISLNK(atributos.st_mode)) strcpy(tipoArchivo,"Enlace relativo
```



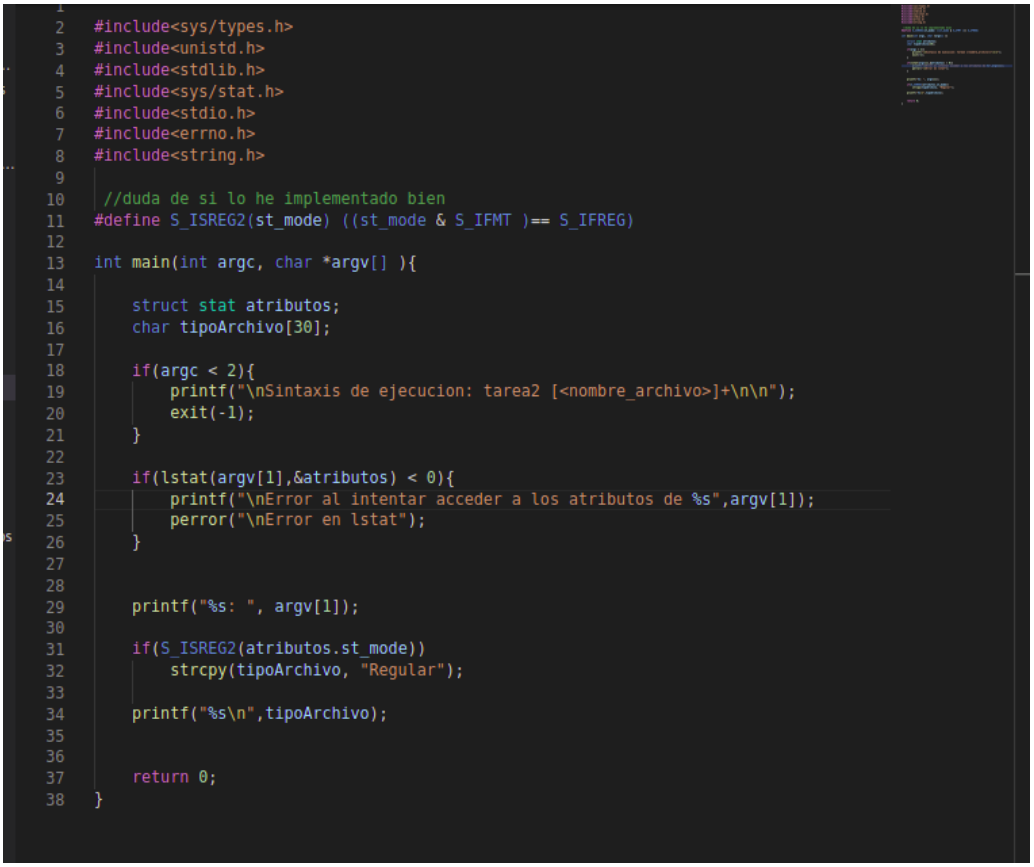
```
(soft)");
     else if(S_ISSOCK(atributos.st_mode)) strcpy(tipoArchivo,"Socket");
     else strcpy(tipoArchivo,"Tipo de archivo desconocido");
     printf("%s\n",tipoArchivo);
   }
}
return 0;
}
```

Este ejercicio, nos indica qué tipo de archivo son los que le pasamos por argumento. Para ello usa las macros de POSIX, y para acceder a ellas crea una estructura de tipo stat, y con ella, accede al campo st_mode de stat, para ver de qué tipo de archivo se trata.

Al ejecutarlo, me daba el siguiente resultado al ejecutarlo, básicamente me indicaba de qué tipo es el archivo salida.txt.



Ejercicio 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 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. #define S_ISREG2(mode) …



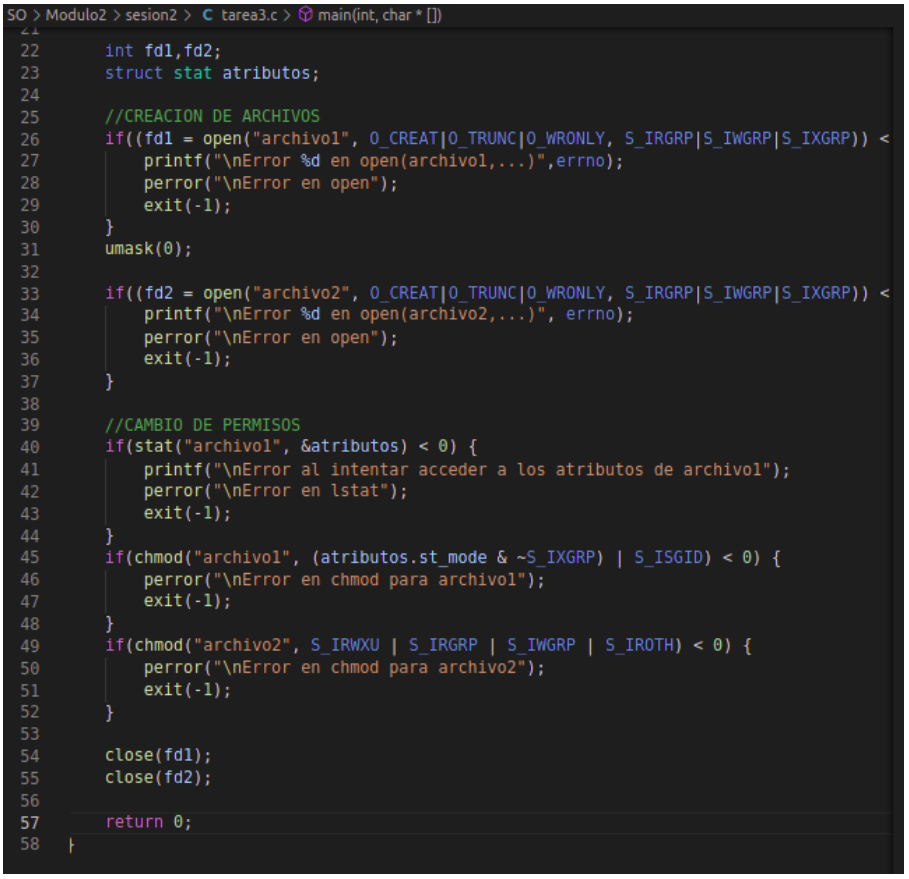
```c
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>

 //duda de si lo he implementado bien
#define S_ISREG2(st_mode) ((st_mode & S_IFMT )== S_IFREG)

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

    struct stat atributos;
    char tipoArchivo[30];

    if(argc < 2){
        printf("\nSintaxis de ejecucion: tarea2 [<nombre_archivo>]+\n\n");
        exit(-1);
    }

    if(lstat(argv[1],&atributos) < 0){
        printf("\nError al intentar acceder a los atributos de %s",argv[1]);
        perror("\nError en lstat");
    }


    printf("%s: ", argv[1]);

    if(S_ISREG2(atributos.st_mode))
        strcpy(tipoArchivo, "Regular");

    printf("%s\n",tipoArchivo);


    return 0;
}
```

Creamos un programa como el anterior, pero en el que sólo se compruebe si el archivo es regular.
Antes del main, se debe definir la nueva MACRO, que en este caso se le llama S_ISREG2.

SESIÓN 2 MÓDULO II SO

Ejercicio 3. ¿Qué hace el siguiente programa?

tarea3.c




```c
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)) <
    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)) <
    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;
}
```

En este programa, se crean 2 archivos, archivo1 y archivo2 y se especifican sus correspondientes errores si no cumplen determinadas condiciones para crearse.

Antes de crearse el segundo archivo se pone la máscara a 0 con umask(0), lo que significa que el archivo2 se creará sin ningún tipo de permisos, ya que si no pusiéramos eso, por defecto tendrán permisos que se dan al crear archivos por defecto.

A continuación, con stat, copiamos el contenido del archivo1 a la estructura stat atributos, para poder acceder a su campo st_mode y poder modificar los permisos.

En el chmod del archivo1, hacemos un and lógico para cambiarle los permisos, negando el permiso de ejecución para el grupo (S_IXGRP), por lo que le quitamos ese permiso, y también le activamos la asignación del GID del propietario al GID efectivo del proceso que ejecute el archivo.

En el chmod del archivo2, cambiamos sus permisos tal que el usuario tenga permisos de lectura, escritura y ejecución; el grupo tenga permiso de lectura; el grupo tenga permiso de escritura; y otros tenga permiso de lectura.

```
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion2$ ls
archivo1  archivo2  tarea3  tarea3.c
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion2$ stat archivo1
  Fichero: archivo1
  Tamaño: 0             Bloques: 0          Bloque E/S: 4096   fichero regular vacío
Dispositivo: 10305h/66309d      Nodo-i: 792877      Enlaces: 1
Acceso: (2060/----rwS---)  Uid: ( 1000/  morris)   Gid: ( 1000/  morris)
Acceso: 2022-12-15 18:27:56.626614322 +0100
Modificación: 2022-12-15 18:27:56.626614322 +0100
      Cambio: 2022-12-15 18:27:56.626614322 +0100
    Creación: -
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion2$ stat archivo2
  Fichero: archivo2
  Tamaño: 0             Bloques: 0          Bloque E/S: 4096   fichero regular vacío
Dispositivo: 10305h/66309d      Nodo-i: 792878      Enlaces: 1
Acceso: (0764/-rwxrw-r--)  Uid: ( 1000/  morris)   Gid: ( 1000/  morris)
Acceso: 2022-12-15 18:27:56.626614322 +0100
Modificación: 2022-12-15 18:27:56.626614322 +0100
```