Sistemas Operativos

Formulario de auto-evaluación

Molulo 2. Sesión 2. Llamadas al sistema para el S.Archivos Parte II

Nombre y apellidos:

Juan Carlos Ruiz García

a) Cuestionario de actitud frente al trabajo.

El tiempo que he dedicado a la preparación de la sesión antes de asistir al laboratorio ha sido de ..... minutos.

1. He resuelto todas las dudas que tenía antes de iniciar la sesión de prácticas: ..... (si/no). En caso de haber contestado "no", indica los motivos por los que no las has resuelto:

Si. En mis horas de practicas pregunté todas las dudas que tenia y ademas, otro dia fuí a una clase de un grupo distinto para solventar algunas dudas que me surgieron acerca de *umask*.

2. Tengo que trabajar algo más los conceptos sobre:

Gracias a consultar las dudas creo que lo llevo todo bastante bien.

3. Comentarios y sugerencias:

Ninguna de momento.

b) Cuestionario de conocimientos adquiridos.

Mi solución al **ejercicio 1** ha sido:

```
#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>
#include<stdlib.h>

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

//CREACION DE ARCHIVOS
```

```
//Crea el fichero archivo1 si no existe con los permisos 070 & ~022 = 050 = ---r-x---
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(EXIT_FAILURE);
}

//Crea el fichero archivo2 si no existe con los permisos 070 & ~000 = 070 = ---rwx---
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(EXIT_FAILURE);
}

//CAMBIO DE PERMISOS
//stat guarda en atributos los permisos del archivo1, en caso de error termina el programa
if(stat("archivo1",&atributos) < 0) {
        printf("\nError al intentar acceder a los atributos de archivo1");
        perror("\nError en lstat");
        exit(EXIT_FAILURE);
}

//Mostrar los permisos del archivo 1 antes de realizar su modificacion.
printf("File Permissions of archivo 1: \t");
 printf( (S_ISDIR(atributos.st_mode)) ? "d" : "-");
 printf( (atributos.st_mode & S_IRUSR) ? "r" : "-");
 printf( (atributos.st_mode & S_IWUSR) ? "w" : "-");
 printf( (atributos.st_mode & S_IXUSR) ? "x" : "-");
 printf( (atributos.st_mode & S_IRGRP) ? "r" : "-");
 printf( (atributos.st_mode & S_IWGRP) ? "w" : "-");
 printf( (atributos.st_mode & S_IXGRP) ? "x" : "-");
 printf( (atributos.st_mode & S_IROTH) ? "r" : "-");
 printf( (atributos.st_mode & S_IWOTH) ? "w" : "-");
 printf( (atributos.st_mode & S_IXOTH) ? "x" : "-");
 printf("\n\n");

// Cambia los permisos de archivo1 de  a (050 - 010) + 2000 = 2040 = ---r-s---
if(chmod("archivo1", (atributos.st_mode & ~S_IXGRP) | S_ISGID) < 0) {
        perror("\nError en chmod para archivo1");
        exit(EXIT_FAILURE);
}

//Cambia los permisos del archivo2 de 070 a (700 + 040 + 020 + 004) = 764 ) rwxrw-r--
if(chmod("archivo2",S_IRWXU | S_IRGRP | S_IWGRP | S_IROTH) < 0) {
        perror("\nError en chmod para archivo2");
        exit(EXIT_FAILURE);
}

return EXIT_SUCCESS;
}
```

Mi solución a la **ejercicio 2** ha sido:

Adjunto llamado ejer2.c (De todas formas se lo pongo a continuación en texto).

```
//Nombre: Juan Carlos Ruiz García             Grupo: C2              Año: 2016
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <dirent.h> //opendir
#include <stdio.h> //strlen y strtol
#include <string.h> //strcmp

//Muestra los permisos de un struct stat como los muestra un ls -l para que sea mas legible
void showPermission(struct stat *atributos){
  printf( (atributos->st_mode & S_IRUSR) ? "r" : "-");
  printf( (atributos->st_mode & S_IWUSR) ? "w" : "-");
```

```
  printf( (atributos->st_mode & S_IXUSR) ? "x" : "-");
  printf( (atributos->st_mode & S_IRGRP) ? "r" : "-");
  printf( (atributos->st_mode & S_IWGRP) ? "w" : "-");
  printf( (atributos->st_mode & S_IXGRP) ? "x" : "-");
  printf( (atributos->st_mode & S_IROTH) ? "r" : "-");
  printf( (atributos->st_mode & S_IWOTH) ? "w" : "-");
  printf( (atributos->st_mode & S_IXOTH) ? "x" : "-");
}

int main(int argc, char const *argv[]) {
 struct stat atr_dir;
 unsigned int permis;
 DIR *dir;
 struct dirent *mi_dirent;

 //Compruebo que el numero de parametros sea el correcto
 if( argc != 3 ){
  printf("Numero de parametros incorrecto.\n");
  printf("\tSintaxis: %s <pathname> <permisos_en_octal_de_4_digitos>\n",argv[0]);
  printf("\tEjemplo: %s /home/juanka1995/ 0775\n",argv[0]);
  exit(EXIT_FAILURE);
 }

 // Almaceno los permisos del directorio pasado como primer parametro
 if(stat(argv[1],&atr_dir) < 0) {
        printf("Error al intentar acceder a los atributos de %s, podria no ser un directorio\n",argv[1]);
        perror("Error en lstat");
        exit(EXIT_FAILURE);
 }

 //Compruebo que el primer parametro pasado sea realmente un directorio
 if( (atr_dir.st_mode & S_IFMT) != S_IFDIR){
  printf("\nError %s no es un directorio.",argv[1]);
  exit(EXIT_FAILURE);
 }

 //Compruebo que el numero pasado sea un numero valido para ser octal
 if( strlen(argv[2]) != 4 || argv[2][0] != '0'){
  printf("El numero %s no es exactamente de 4 digitos comenzando por 0\n",argv[2]);
  exit(EXIT_FAILURE);
 }

 //Paso el *char a integer para asignar permisos posteriormente.
 permis = strtol(argv[2],NULL,8);

 //Abrimos el directorio pasado por argumento
 dir = opendir(argv[1]);
 if(dir == NULL){
  printf("Error. No se puede abrir el directorio %s\n",argv[1]);
  exit(EXIT_FAILURE);
 }

 printf("Si todo va bien la 2ª columna mostrara <p_antiguos> y la 3ª mostrara <p_nuevos>\n");
 printf("Si falla la 2ª columna mostrara <error> y la 3ª mostrara p_antiguos>\n\n");
 printf("<nombre_archivo>\t\t\t<p_antiguos/error>\t\t<p_nuevos/antiguos>\n\n");
  //Mientras existan ficheros/directorios en el directorio abierto vamos avanzando seleccionando uno
distinto en cada iteracion
 while ((mi_dirent = readdir(dir)) != NULL){
  // Guardamos el nombre del fichero/directorio seleccionado
  char *dname = mi_dirent->d_name;
  // Evitamos localizar . y ..
  if(strcmp(dname,".") && strcmp(dname,"..")){
   //Creamos un vector de char que contenga el path completo del fichero/directorio seleccionado
   char dirp[strlen(argv[1]+strlen(dname))];
   strcpy(dirp,argv[1]);
   strcat(dirp,dname);
   //Guardamos los permisos de dicho fichero/directorio en atr_dir
   stat(dirp,&atr_dir);
   //Si es un fichero regular entramos en el if
   if((atr_dir.st_mode & S_IFMT) == S_IFREG){
    //Si al intentar dar permisos diera error mostrariamos lo siguiente
    //NOTA : el chmod en el if solo devuelve el resultado, es decir se pueden cambiar los permisos
    // o no se pueden pero no los llega a cambiar.
    if(chmod(dirp,permis) < 0){ //Muestra <nombre_archivo> <error> <p_antiguos>
```

```
     printf("%s\t\t%s\t\t",dirp,strerror(errno));
     showPermission(&atr_dir);
     printf("\n");
    }
    else{ //Muestra <nombre_archivo> <p_antiguos> <p_nuevos>
     printf("%s\t\t",dirp);
     showPermission(&atr_dir);
     printf("\t\t\t");
     //Cambiamos los permisos
     chmod(dirp,permis);
     stat(dirp,&atr_dir);
     showPermission(&atr_dir);
     printf("\n");
    }
   }
  }
 }

 //Cierra el directorio abierto
 closedir(dir);

 return 0;
}
```

Mi solución a la **ejercicio 3** ha sido:

Adjunto llamado ejer3.c (De todas formas se lo pongo a continuación en texto).

```
//Nombre: Juan Carlos Ruiz García              Grupo: C2              Año: 2016
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <dirent.h> //opendir
#include <stdio.h> //strlen y strtol
#include <string.h> //strcmp

off_t full_size = 0; //Almacenara el tamaño total en bytes de todos los archivos regulares encontrados
int n_files = 0; //Almacenara el numero de archivos encontrados
const unsigned int D_PERMISOS = 0011; // Permisos de ejecucion para grupo y otros

// Busca de forma recursiva dentro de un directorio y sus subdirectorios, archivos regulares
// con permisos de ejecucion en el grupo y en otros.
void buscar(char * pathname){
 DIR *dir;
 struct dirent *mi_dirent;
 char * next_path;
 struct stat atr_dir;
 //Abrimos el directorio pasado como parametro de la funcion buscar
 dir = opendir(pathname);
 if(dir == NULL){
  printf("\t\tError. No se puede abrir el directorio %s\n",pathname);
 }
 else{
   //Mientras existan ficheros/directorios en el directorio abierto vamos avanzando seleccionando uno
distinto en cada iteracion
  while ((mi_dirent = readdir(dir)) != NULL){
   // Guardamos el nombre del fichero/directorio seleccionado
   char *dname = mi_dirent->d_name;
   // Evitamos localizar . y ..
   if(strcmp(dname,".") && strcmp(dname,"..")){
    //Creamos un vector de char que contenga el path completo del fichero/directorio seleccionado
    char dirp[strlen(pathname)+strlen(dname)];
    strcpy(dirp,pathname);
    //Si el pathname no acaba en / se la añadimos
    if(pathname[strlen(pathname)-1] != '/')
     strcat(dirp,"/");
    strcat(dirp,dname);
    //Guardamos los permisos de dicho fichero/directorio en atr_dir
```

```
    stat(dirp,&atr_dir);
    /*
    Si es un directorio llamamos recursivamente a la misma funcion buscar para
    que busque dentro de dicho directorio archivos regulares con permisos de
    ejecucion para grupo y otros.

    NOTA : Para evitar problemas he ignorado todos los directorios ocultos,
    es decir los que empiezan por .<nombre_directorio>
    */
    if( ((atr_dir.st_mode & S_IFMT) == S_IFDIR) && dname[0] !='.' ){
     buscar(dirp);
    }
    // Si no es un directorio y es un archivo regular...
    else if ( (atr_dir.st_mode & S_IFMT) == S_IFREG){
     // Si es un archivo regular y tiene al menos los permisos 011 = -----x--x,
     // muestro la ruta absoluta del archivo, su numero de inodo, incremento el contador
     // y sumo el tamaño de dicho archivo regular a la variable full_size
     if ((atr_dir.st_mode & D_PERMISOS) == D_PERMISOS){
      printf("\t%s %lu\n",dirp,atr_dir.st_ino);
      n_files++;
      full_size += atr_dir.st_size;
     }
    }
   }
  }
 }
}

int main(int argc, char const *argv[]) {
 struct stat atr_dir;
 char *pathname = malloc(sizeof(char) * 254); // assuming the max length of a string is not more than 253
characters

 if(argc <= 2){
  if(argc == 2)
   strcpy(pathname,argv[1]);
  else
   pathname = "./";

  // Almaceno los permisos del directorio pasado como primer parametro
  if(stat(pathname,&atr_dir) < 0) {
   printf("Error al intentar acceder a los atributos de %s, podria no ser un directorio\n",argv[1]);
   perror("Error en stat");
   exit(EXIT_FAILURE);
  }

  //Compruebo que el primer parametro pasado sea realmente un directorio
  if( (atr_dir.st_mode & S_IFMT) != S_IFDIR){
   printf("Error %s no es un directorio.\n",argv[1]);
   exit(EXIT_FAILURE);
  }

  printf("Los inodos son: \n\n");
  buscar(pathname);
     printf("\nExisten %d archivos regulares con permisos x para grupo y otros en el directorio
%s\n",n_files,pathname);
      printf("El tamaño total ocupado por dichos archivos es %ld Bytes = %ld MB aprox.\n",
full_size,full_size/(1024*1024));
 }
 else{
  printf("Numero de parametros incorrecto\n");
  printf("\tFormato: %s <directorio_a_buscar>\n",argv[0]);
  printf("NOTA : Si no se especifica <directorio_a_buscar> se utilizar ./ por defecto\n");
 }
 return 0;
}
```