Examen-SO-Practicas-Resuelto.pdf

Zukii

Sistemas Operativos

2º Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

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

@Zukii on Wuolah

Examen Sistemas Operativos Prácticas

Ejercicio 1 [4 puntos]

Desarrolle un programa en C (ejercicio1.c) utilizando llamadas al sistema y haciendo uso de los mecanismos que considere oportunos. El programa deberá copiar los archivos regulares con tamaño superior o igual a 1 KB de un directorio que será el primer argumento (deberá recorrer la jerarquía de subdirectorios existentes) a otro directorio (segundo argumento). Si no existe el segundo directorio debe de crearse.

Por cada archivo copiado, escribir una línea de caracteres en un fichero con nombre copiados.txt siguiendo el formato.

Nombredelarchivo|permisos|número de inodo

y al final la suma total de espacio ocupado por todos los archivos.

Un ejemplo de archivo de salida sería:

ejercicio.txt|677|12

datos.txt|777|141

Tamaño total:213678 bytes

NOTA: En la cabecera del programa debe aparecer (en comentarios) el nombre y los apellidos del autor. No olvides comentar tu código y realizar el tratamiento de errores adecuado.

Deben enviarse el archivo del programa y capturas de pantalla completa(no recortéis la captura) donde se vea la ejecución del programa en tu entorno.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
```

@Zukii on Wuolah

```
#include <string.h>
#include <dirent.h>
#include <errno.h>

//Álvaro Vega Romero
const int BYTES_EN_KB = 1024;
int tam = 0; //EN bytes

void copiar(DIR *dir, char *camino, DIR *otro_dir, char * otro_camino); //Se pasa los punt a los dir y pathnames

int main(int argc, char *argv[])
{
	DIR *dir; //puntero a directorio original
	DIR *otro_dir;

	if (argc != 3)
	{
		printf("Número de argumentos erroneo, introduzca 3\n");
	}
	else
	{
		dir = opendir(argv[1]); //puntero a directorio

		if(dir == NULL)
		{
			printf("Error al abrir el directorio pasado como primer parámetro\n");
		}

		otro_dir = opendir(argv[2]); //puntero al toro directorio
```

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

```
                if(otro_dir == NULL)
                {
                        mkdir(argv[2], 777);
                        otro_dir = opendir(argv[2]);
                }

                copiar(dir, argv[1], otro_dir, argv[2]);

                int fd;

                if(
(fd=open("copiado.txt",O_CREAT|O_TRUNC|O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP))<0)
                {
                        printf("\nError %d en open de copiados.txt\n",errno);
                        perror("\nError en open\n");
                        exit(EXIT_FAILURE);
                }

                char cadena_copiados[64];

                sprintf(cadena_copiados, "Tamaño total: %d\n" ,tam);
                write(fd, cadena_copiados, sizeof(cadena_copiados));

                closedir(dir); //Cerramos el directorio
                closedir(otro_dir); //Cerramos el directorio
        }
}

void copiar(DIR *dir, char *camino, DIR *otro_dir, char * otro_camino)
{
        struct dirent *entrada; //Devuelve la entrada a través de un puntero a la estructura
dirent
```

```
        struct stat atributos; //Los metadatos de un archivo
        char pathname[512]; //Para guardar una cadena
        DIR *subdir; //puntero a sub-directorio de dir
        char cadena_copiados[512];

        int fd;



        if( (fd=open("copiado.txt",O_CREAT|O_TRUNC|O_WRONLY,777)<0))
        {
                printf("\nError %d en open de copiados.txt\n",errno);
                perror("\nError en open\n");
                exit(EXIT_FAILURE);
        }

        while((entrada = readdir(dir)) != 0) //Lee donde esta el puntero dir.
        {
                if (strcmp(entrada->d_name, ".") && strcmp(entrada->d_name, ".."))
//Comprobamos si no son los ficheros . y ..
                {
                        sprintf(pathname, "%s/%s", camino, entrada->d_name); //Guarda en
pathname el string. LA ruta
                        lstat(pathname, &atributos); //  stat examina el fichero al que apunta
pathname y llena buf. lstat con enlace simbólico lo examina


                        if (S_ISREG(atributos.st_mode) && (atributos.st_size >=
BYTES_EN_KB))
                        //Archivo regular y que el tamaño sea mayor a 1KB
                        {
                                sprintf(cadena_copiados, "%s|%i|%li",entrada
                                ->d_name,atributos.st_mode, atributos.st_ino);
```

@Zukii on Wuolah

```
                write(fd, cadena_copiados, sizeof(cadena_copiados));
                tam += atributos.st_size;          //Aumentamos

                //Copiar

                execlp("cp",pathname,otro_camino,NULL);
            }

            else if (S_ISDIR(atributos.st_mode)) //SI es un directorio
            {
                subdir = opendir(pathname); //Repetimos el proceso anterior(recursiva)
                copiar(subdir, pathname, otro_dir, otro_camino);
        //Buscamos en el subdirectorio
                closedir(subdir); //Cerramos el directorio
            }
        }
    }
    close(fd);
}
```

Ejercicio 2 [6 puntos]

Crear un segundo programa en C (ejercicio2.c) que se ejecute en un bucle infinito de tal forma que lea cadenas de texto hasta que reciba la palabra "fin". Este programa debe de crear dos hijos, y se debe comunicar con ellos a través del mecanismo de comunicación que desee. Cuando el proceso padre reciba una cadena de texto debe de comprobar si el archivo con dicho nombre existe o no. Si no existe se creará.

El padre enviará el número a los dos hijos que realizarán los siguientes procesos:

-El primer hijo determinará si el número de inodo del archivo recibido por el padre como argumento es par o impar, y sacará un mensaje por pantalla del estilo: “Soy el hijo 1 y el inodo es par”.

-El segundo hijo realizará un cambio de permisos del archivo a 666.

Consideraciones:

-Se debe tener en cuenta toda la gestión de errores posible.

-No deben quedar procesos zombies (y si quieres ser compasivo/a deja zombis vivos, pero pon un comentario de cómo acabar con ellos).

-En la cabecera del programa debe aparecer (en comentarios) el nombre y los apellidos del autor. No olvides comentar tu código

-Deben enviarse el archivo del programa y capturas de pantalla completa(no recortéis la captura) donde se vea la ejecución del programa en tu entorno

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
```

@Zukii on Wuolah

```
#include <string.h>
#include <dirent.h>
#include <errno.h>

//Álvaro Vega Romero

//Para acabar con los procesos zombies, lo mejor que podemos hacer es que cuando se reciba una señal de terminacion de hijo
//tener un manejador tal que asi para saber que ha terminado y evitar que se queden en estado zombie
/*
void handler()
{
	int estado;
	pid_t pid;

	pid = wait(&estado);

	printf("Mi hijo %d ha finalizado con estado %d\n", pid, estado);

	exit(EXIT_SUCCESS);
}
*/

//Ya que no me da tiempo a acabar correctamente el ejercicio, lo que habría que retocar es la comprobacion
// de si la palabra recibida es fin, ya que no sé por qué genera un bucle infinito.
//Tambien destacar que al usar chmod no sale ningun mensaje por pantalla a no ser que haya error

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

```
	pid_t PID1;
	pid_t PID2;
	char archivo[256];

	//Descriptores de hijos y padre
	int fdE1[2], fdE2[2], fdM[2];

	//Cauces
	pipe(fdE1); //hijo1
	pipe(fdE2); //hijo2
	pipe(fdM); //padre


	if((PID1 = fork()) == 0) //hijo1 - recibirá el num inodo;
	{
		int inodo;

		close(fdE2[0]); //No interactua con el hijo 2
		close(fdE2[1]);

		close(fdE1[1]); //El hijo escribe en la salida estandar - No en el cauce
		close(fdM[1]); //Lee del padre - Cerramos escritura

		dup2(fdE1[0], STDIN_FILENO); //duplica el fd de lectura sobreescribiendo
entrada estandar

		while(1)
		{
			read(STDIN_FILENO, &inodo, sizeof(inodo)); //Hemos sobrecargado la
entradaestandar

			if(inodo % 2 == 0) //es par
```

@Zukii on Wuolah

```
                {
                        printf("Soy el hijo 1 y el num de inodo %i, es par\n", inodo);
                }

                else
                {
                        printf("Soy el hijo 1 y el num de inodo %i, NO es par\n", inodo);
                }
        }
}

else if(PID1 < 0) //SI fork da error
{
        perror("fork");
        exit(EXIT_FAILURE);
}

if((PID2 = fork()) == 0) //hijo2
{
        char archivo_h[256];

        close(fdE1[0]); //No interactua con el hijo 1
        close(fdE1[1]);

        close(fdM[1]); //Lee del padre - Cerramos escritura
        close(fdE1[1]); //El hijo no escribe y si escribiese sería en la salida estandar NO en el cauce

        dup2(fdE2[0], STDIN_FILENO); //duplica el fd de lectura sobreescribiendo entrada estandar

        while(1)
```

```
		{
			read(STDIN_FILENO, &archivo_h, sizeof(archivo_h));

			if(chmod(archivo_h, 666) < 0)
			{
				printf("Error al aplicar chmod al archivo %s\n", archivo_h);
			}
		}

		//SI funciona, no decimos nada
	}

	else if(PID1< 0)
	{
		perror("fork");
		exit(EXIT_FAILURE);
	}

	else //Padre
	{

		int inodo;
		struct stat atributos;

		while(1)
		{
			int bytes;
			bytes = read(STDIN_FILENO, &archivo, sizeof(archivo)); //leemos ruta
desde entrada estandar

/*
```

```
                        if(strcmp(archivo, "fin")) //SI la cadena recibida es fin
                        {
                                printf("Terminando...\n");
                                exit(EXIT_SUCCESS);
                        }
*/
                        if(1==1) //Si no es - Deberia ser un else
                        {
                                int fd;

                                //Comprobar si archivo existe y si no existe crear
        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(EXIT_FAILURE);
                                }

                                lstat(archivo, &atributos); //Ver sus atributos

                                write(fdE1[1], &(atributos.st_ino) , sizeof(int)); //Enviamos
inodo al hijo 1
                                write(fdE2[1], &archivo, bytes); //Enviamos ruta al hijo 2

                        }
                }
        }
}
```

Cabe decir que la nota fue de 7,25 sobre 10. El primer ejercicio no funciona y viene a partir del boceto del ejercicio 3 de la sesión 2 y el segundo ejercicio la función strcmp crea un bucle infinito y por eso hago el if(1==1)