# Sistemas Operativos

## Formulario de auto-evaluación

### Modulo 2. Sesión 6. Control de archivos y archivos proyectados en memoria

**Nombre y apellidos:**

| |
|---|
| Ana Alicia Vílchez Ceballos |

**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 120 minutos.

1. He resuelto todas las dudas que tenía antes de iniciar la sesión de prácticas: no. En caso de haber contestado “no”, indica los motivos por los que no las has resuelto:

| |
|---|
| No exáctamente. Durante la realización de los ejercicios me han surgido dudas que he de aclarar |

2. Tengo que trabajar algo más los conceptos sobre:

| |
|---|
| Proyección de archivos |

3. Comentarios y sugerencias:

| |
|---|
| |

**b) Cuestionario de conocimientos adquiridos.**

Mi solución a la **ejercicio 1** ha sido:

```
// ejercicio1.c
/*
Ejercicio 1. Implementa un programa que admita t argumentos. El primer argumento será una
orden de Linux; el segundo, uno de los siguientes caracteres “<” o “>”, y el tercero el nombre de
un archivo (que puede existir o no). El programa ejecutará la orden que se especifica como
argumento primero e implementará la redirección especificada por el segundo argumento hacia
el archivo indicado en el tercer argumento.
*/
// precondición: cuando le pasemos el segundo argumento, es decir, > o < hay que ponerlo
// entrecomillado ya que si no, lo toma como redirección.

#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char * argv[]){



        if(argc < 4){
                printf("Error en el numero de argumentos\n");
                printf("pasar como argumentos: orden </> archivo\n");
                exit(-1);
        }

        if(strcmp(argv[2],"<") != 0 && strcmp(argv[2],">") != 0){
                printf("Error en el segundo parámetro\n");
                exit(2);
```

```
	}


	int fd ;
	if((fd = open(argv[3],O_CREAT | O_WRONLY, S_IRWXU)) < 0){
		printf("error al abrir el archivo\n");
		exit(1);
	}


	if(strcmp(argv[2],">") == 0){
		close(1); // cerramos la salida estandar para llevar a cabo la duplicacion


		if(fcntl(fd, F_DUPFD,1) == -1){
			perror("fallo en fcntl");
			exit(1);
		}
		char buffer[256];
		execlp(argv[1], argv[1], NULL); // escribe en el archivo que se ha abierto


		int cont = write(1,buffer,256);
	}
	// PARA ESTE CASO LA FUNCION EXECLP NO COGE LA ENTRADA ESTANDAR QUE HEMOS DUPLICADO, POR QUÉ?
	// Para este caso, el fichero si tiene algo escrito, será un argumento del comando
	else{
		// En este caso lo que tenemos que hacer es cerrar la entrada estándar para que el proceso
		// pueda leer del archivo y así tomar el contenido del archivo como argumento para el comando
		// que le pasamos como primer parámetro
		close(0); // cerramos la entrada estandar para llevar a cabo la duplicacion


		if(fcntl(fd, F_DUPFD,0) == -1){ // duplicamos el descriptor en el "hueco" de la entrada estándar
```

```
                perror("fallo en fcntl");
                exit(1);
        }

        execlp(argv[1], "",NULL);

    }
    return 0;



}
```

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

```
/*
Ejercicio que demuestra que el kernel es capaz de detectar y solucionar el interbloqueo.
Lanzamos el programa desde varias terminales para llevar a cabo varios procesos
sobre el mismo archivo
*/

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

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

int main (int argc, char * argv[])
{
        if(argc < 2){
                printf("Error en el numero de argumentos\n");
                exit(1);
        }

        int fd;

        // abrimos el archivo
        if ((fd = open(argv[1], O_RDWR, S_IRWXU)) == -1 ){
                perror(" fallo en open \n");
                exit(1);
        }

        /* intentamos un bloqueo de escritura  archivo */
        struct flock cerrojo;
        cerrojo.l_type=F_WRLCK;
        cerrojo.l_whence=SEEK_SET;
        cerrojo.l_start=0;
        cerrojo.l_len=0;

        printf("tratando de bloquear el archivo...\n");
        if((fcntl (fd, F_SETLKW, &cerrojo) )== -1) {
                if(errno == EDEADLK)
                    printf("ATENCION: interbloqueo detectado \n");
        }
```

```
        printf("El cerrojo puesto sobre %s ha tenido exito \n", argv[1]);


        sleep(10);
        return 0;
}
```

Mi solución a la **ejercicio 5** ha sido:

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

int main (int argc, char *argv[])
{
      struct stat sb; // seran los atributos del archivo a copiar
      off_t len;
      char *p, *p_d;
      int fd1, fd2;

      if (argc != 3) {
```

```
        printf("Uso: %s archivo_a_copiar archivo_copiado\n", argv[0]);
        exit(1);
}

fd1 = open (argv[1], O_RDONLY,  S_IRWXU);
if (fd1 == -1) {
        printf("Error al abrir archivo\n");
        return 1;
}
fd2 = open (argv[2], O_CREAT | O_RDWR,  S_IRWXU);
if (fd1 == -1) {
        printf("Error al abrir archivo\n");
        return 1;
}

if (fstat (fd1, &sb) == -1) {
        printf("Error al hacer stat\n");
        return 1;
}

if (!S_ISREG (sb.st_mode)) {
        printf ("%s : no es un archivo regular\n", argv[1]);
        return 1;
}

//asignamos espacio al fichero de destio
ftruncate(fd2, sb.st_size);
// llevamos a cabo la proyeccion
p = (char *) mmap(0, sb.st_size, PROT_READ, MAP_SHARED, fd1, 0);
if(p == MAP_FAILED) {
         perror("Error en la proyeccion");
```

```
            exit(2);
        }
        // llevamos a cabo la proyeccion de escritura en el fichero de destino
           p_d = (char *) mmap(0, sb.st_size, PROT_WRITE, MAP_SHARED, fd2, 0);
        if(p_d == MAP_FAILED) {
            perror("Fallo en la segunda proyeccion");
            exit(2);
        }
        //llevamos a cabo la copia
        memcpy(p_d, p, sb.st_size);
        // cerramos las proyecciones
        if (munmap (p, sb.st_size) == -1) {
            printf("Error al cerrar la proyeccion \n");
            return 1;
        }
         if (munmap (p_d, sb.st_size) == -1) {
            printf("Error al cerrar la proyeccion \n");
            return 1;
        }
        return 0;
}
```