Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Jesús Manuel García Palma

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 2 h minutos.

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

2. Tengo que trabajar algo más los conceptos sobre:

3. Comentarios y sugerencias:

b) Cuestionario de conocimientos adquiridos.

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

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

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

	//Comprobamos si se le ha pasado un pathname y unos permisos como parámetros
	if(argc<3) {
		//Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
		printf("Modo de uso: %s <programa> <simbolo> <archivo>\n\n", argv[0]);
		exit(1);
	} else {

		//Declaracion de variables
		char *str_command;
		char *str_file;
		int fd;

		//Extraemos el comando
		str_command = argv[1];
		str_file = argv[3];

		//Comprobamos el segundo parametro, tiene que ser < o >
		if (strcmp(argv[2], "<") == 0) {

			//Redireccion de entrada
```

```
            fd = open (str_file, O_RDONLY);
            close(STDIN_FILENO);
            if (fcntl(fd, F_DUPFD, STDIN_FILENO) == -1 ) perror ("fcntl falló");

        } else if (strcmp(argv[2], ">") == 0) {

            //Redireccion de salida
            fd = open (str_file, O_CREAT|O_WRONLY);
            close (STDOUT_FILENO);
            if (fcntl(fd, F_DUPFD, STDOUT_FILENO) == -1 ) perror ("fcntl falló");


        } else {

            printf("Debe pasarse \"<\" o \">\" con las comillas %s\n\n", argv[2]);

            //Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
            printf("Modo de uso: %s <programa> [opciones] [bg]\n\n", argv[0]);
            exit(1);

        }

        //Ejecutamos el comando
    if( (execlp(str_command, "", NULL) < 0)) {

            perror("Error en el execlp\n");
            exit(-1);
        }

        //Cerramos el fichero
        close(fd);
```

```
	}

}
```

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

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

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

	//Declaracion de variables
	struct flock cerrojo;
	int fd, i;

	if(argc != 2) {

		//Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
		printf("Modo de uso: %s <programa> <archivo>\n\n", argv[0]);
		exit(1);
```

```
} else {

        //Extraemos el nombre del archivo a usar (por comodidad)
        char *str_file = argv[1];

        //Abrimos el archivo
        if ((fd=open(str_file, O_RDWR)) == -1 ){
                perror("Fallo al abrir el archivo");
                return 0;
        }

        cerrojo.l_type=F_WRLCK;
        cerrojo.l_whence=SEEK_SET;
        cerrojo.l_start=0;
        cerrojo.l_len=0; //Bloquearemos el archivo entero

        //Intentamos un bloqueo de escritura del archivo
        printf ("Intentando bloquear %s\n", str_file);
        if (fcntl (fd, F_SETLKW, &cerrojo) == EDEADLK) {

                //Si el cerrojo falla, pintamos un mensaje
                printf ("%s ha dado un EDEADLK\n", str_file);

        } //Mientras el bloqueo no tenga exito

        //Ahora el bloqueo tiene exito y podemos procesar el archivo
        printf ("Procesando el archivo %s\n", str_file);

        //Hacemos un bucle con sleep para que de tiempo a lanzar otra vez el programa
        for (i = 0; i < 10; i++) {
                sleep(1);
```

```
            }
            //Una vez finalizado el trabajo, desbloqueamos el archivo
            cerrojo.l_type=F_UNLCK;
            cerrojo.l_whence=SEEK_SET;
            cerrojo.l_start=0;
            cerrojo.l_len=0;
            if (fcntl (fd, F_SETLKW, &cerrojo) == -1) {
                  perror ("Error al desbloquear el archivo");
            }

            return 0;
      }
}
```

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

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

```
#include <fcntl.h>

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

	//Comprobamos si se le ha pasado un pathname y unos permisos como parámetros
	if(argc != 3) {
		//Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
		printf("Modo de uso: %s <programa> <origen> <destino>\n\n", argv[0]);
		exit(1);

	} else {

		//Declaracion de variables
	struct stat sb;
		char *str_orig = argv[1];
		char *str_dest = argv[2];



		int fd_orig, fd_dest;
		char *mem_orig, *mem_dest;
		int filesize;



		//Abrimos el fichero de origen
		fd_orig = open(str_orig, O_RDONLY);
		if (fd_orig == -1) {
			perror("Fallo al abrir el archivo de origen\n");
			exit(2);
		}

		//Obtenemos su stat, para comprobar si es regular y obtener su tamaño
```

```
if (fstat (fd_orig, &sb) == -1) {
        printf("Error al hacer stat en el fichero de origen\n");
        return 1;
}

if (!S_ISREG (sb.st_mode)) {
        printf ("El fichero de origen no es un archivo regular\n");
        return 1;
}

//Guardamos el tamaño en una variable (por comodidad)
filesize = sb.st_size;

//Creamos el archivo de destino
umask(0);
        fd_dest = open(str_dest, O_RDWR|O_CREAT|O_EXCL, S_IRWXU);
        if (fd_dest == -1) {
                perror("Fallo al crear el archivo de salida");
                exit(2);
        }

        //Asignamos el espacion en el fichero de destino
        ftruncate(fd_dest, filesize);

        //Creamos el mapa de memoria del fichero de origen
        mem_orig = (char *) mmap(0, filesize, PROT_READ, MAP_SHARED, fd_orig, 0);
        if(mem_orig == MAP_FAILED) {

                perror("Fallo mapeando el archivo de entrada");
                exit(2);
```

```
            }

            //Creamos el mapa de memoria del fichero de destino
            mem_dest = (char *) mmap(0, filesize, PROT_WRITE, MAP_SHARED, fd_dest, 0);
            if(mem_dest == MAP_FAILED) {

                perror("Fallo mapeando el archivo de salida");
                exit(2);

            }

            //Copiamos un mapa de memoria en otro
            memcpy(mem_dest, mem_orig, filesize);

            //Liberamos los mapas de memoria
            munmap(mem_orig, filesize);
            munmap(mem_dest, filesize);

            //Cerramos los descriptores de fichero
            close(fd_orig);
            close(fd_dest);

            //Terminamos la ejecución del programa
            return 0;
        }
}
```