Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Antonio Miguel Morillo Chica

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 0 minutos.

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

Faltar a clase.

2. Tengo que trabajar algo más los conceptos sobre:

Toda esta practica total.

3. Comentarios y sugerencias:

Ninguno.

b) Cuestionario de conocimientos adquiridos.

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

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

int main(int argc, char *argv[]){
	if (argc < 4)
		 printf("Error falta de argumentos: ./a.out progrmama redireccion temporal\n");

	int fd;
	const char * ordenLinux=argv[1];
	char * caracteres=argv[2];
	char * nombreArchivo=argv[3];

	// Abro el archivo para leer/escribir
	if((fd=open(nombreArchivo,O_RDWR|O_TRUNC,1))<0) {
		perror("\nError en open");
		exit(-1);
	}
	// Lectura
	if (strcmp(caracteres,"<")==0) {
		// Cierro la entrada estandar para cambiarla por el archivo
		close(STDIN_FILENO);
		// Duplico el descriptor.
		if (fcntl(fd, F_DUPFD,STDIN_FILENO)==-1) {
			perror("fcntl fallo");
			exit(-1);
		}
	}
	// Escritura
	if (strcmp(caracteres,">")==0) {
		// Cierro la salida estandar para cambiarla por el archivo
		close(STDOUT_FILENO);
		// Duplico el descriptor.
		if (fcntl(fd, F_DUPFD,STDOUT_FILENO)==-1) {
			perror("Error: fcntl fallo\n");
			exit(-1);
		}
	}
	// Ejecutamos la ordenLinux
	if(execlp(ordenLinux,ordenLinux,NULL)<0) {
		perror("Error en el execv\n");
		exit(-1);
	}
	close(fd);
	return(0);
}
```

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

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

int main(int argc, char const *argv[]) {
  // Tipo de cerrojo que he usado
  struct flock cerrojo = {F_WRLCK, SEEK_SET, 0, 0, 0};
  int fd;

  // Cerrojo para este proceso.
  cerrojo.l_pid=getpid();

  // Itero para cada archivo
  for (int i = 1; i < argc; i++) {
    // En cada vuelta abro un archivo
    if ((fd=open(argv[i], O_RDWR)) < 0) {
      printf("Error en abrir el archivo %s", argv[i]);
      exit(1);
    }
    // Mensaje previo.
    printf("Cuando pulses <ENTER> se producirá un bloqueo: \n" );
    getchar();
    printf("Produciendo un bloqueo... \n" );
    int block;
    // Asigno al archivo el cerrojo anterior
    if((block = fcntl(fd, F_SETLK, &cerrojo)) == -1){
        perror("Error en el fcntl, ya hay un cerrojo!\n");
        exit(1);
    }
    // Si ejecuto el programa desde otra pestaña en este punto
    // se producirá un error ya que hay un cerrojo en ese momento activo.
    printf("Bloque producido, estado del block: %i\n", block);
    printf("Cuando pulses <ENTER> se producirá un desbloqueo: \n" );
    getchar();
    // Desbloqueo el cerrojo
    cerrojo.l_type = F_UNLCK;
    // Aplico el cambio
    if((block = fcntl(fd, F_SETLK, &cerrojo)) == -1){
      perror("Error al aplicar el desbloqueo\n");
      exit(1);
    }

    printf("Desbloqueado, estado: %i \n", block);
    close(fd);  // Cierro el archivo
  }

  return 0;
}
```

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

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

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

  if (argc != 3) {
    printf("Modo de uso: ./%s <ruta_origen> <ruta_destino>\n", argv[0]);
    exit(1);
  }

  struct stat atributos;
  const char *origen = argv[1],
             *destino = argv[2];

  int fd_origen, fd_destino;
  char *mem_destino;
  const char *mem_origen;
  int sizefile;

  // Abro el archivo
  if ((fd_origen = open(origen, O_RDONLY)) < 0) {
    perror("Error al abrir open");
    exit(-1);
  }
  // Copio los atributos
  if (fstat(fd_origen, &atributos) < 0) {
    perror("Error al copiar los atriutos\n");
    exit(-1);
  }

  // Guardo el tamño necesario para copiar
  sizefile = atributos.st_size;
  umask(0);

  // Abro el archivo destino y sino existe lo creo
  if ((fd_destino = open(destino, O_RDWR|O_CREAT|O_EXCL,S_IRWXU)) < 0) {
    perror("Error al abrir open");
    exit(-1);
  }
```

```
  // Igualo el espacio necesario de memoria
  ftruncate(fd_destino, sizefile);

  // Creo el mapa de memoria del primer archivo
   if ((mem_origen = ((char *) mmap(0, sizefile, PROT_READ, MAP_SHARED, fd_origen, 0))) ==
MAP_FAILED) {
    perror("Fallo al mapear la memoria\n");
    exit(-1);
  }
  // Creo el mapa de memoria del segundo archivo
  if ((mem_destino = ((char *) mmap(0, sizefile, PROT_WRITE, MAP_SHARED, fd_destino, 0))) ==
MAP_FAILED) {
    perror("Fallo al mapear la memoria\n");
    exit(-1);
  }

  // Copio el mapa origen en destino.
  memcpy(mem_destino, mem_origen, sizefile);

  // Cierro los archivos.
  close(fd_destino);
  close(fd_origen);

  return 0;
}
```