SESIÓN 6 SO MODULO II

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. Por ejemplo, si deseamos redireccionar la entrada estándar de sort desde un archivo temporal, ejecutaríamos:

$> ./mi_programa sort “<” temporal

Nota. El carácter redirección (<) aparece entre comillas dobles para que no los interprete el shell sino que sea aceptado como un argumento del programa mi_programa.

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

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

    const int T = 4;

    if (argc != T){
        printf("Error en el numero de parametros.");
        exit(EXIT_FAILURE);
    }

    char *orden = argv[1];
    char *caracter = argv[2];
    char *archivo = argv[3];
    int fd;

    //Redireccionamos la entrada o salida estándar dependiendo de si es < o >
    //entrada --> <
    //salida --> >

    //ENTRADA
    if(strcmp(caracter, "<") == 0){
        fd = open(archivo, O_RDONLY);
        close(STDIN_FILENO);

        if(fcntl(fd, F_DUPFD, STDIN_FILENO) == -1)
            perror("\nError en fcntl");
    }

    //SALIDA
    else if (strcmp(caracter, ">") == 0){
        fd = open(archivo, O_CREAT|O_WRONLY);
        close(STDOUT_FILENO);

        if(fcntl(fd, F_DUPFD, STDOUT_FILENO) == -1)
            perror("\nError en fcntl");
    }

    else{
        printf("\nEl segundo parametro debe ser uno de los anteriores");
    }

    //ejecuta la orden
    if (execlp(orden, "", NULL) < 0){
        perror("Error en excelp\n");
        exit(EXIT_FAILURE);
    }
}
```

Ejercicio 2. Reescribir el programa que implemente un encauzamiento de dos órdenes pero utilizando fcntl. Este programa admitirá tres argumentos. El primer argumento y el tercero serán dos órdenes de Linux. El segundo argumento será el carácter “|”. El programa deberá ahora hacer la redirección de la salida de la orden indicada por el primer argumento hacia el cauce, y redireccionar la entrada estándar de la segunda orden desde el cauce.
Por ejemplo, para simular el encauzamiento ls|sort, ejecutaríamos nuestro programa como:

$> ./mi_programa2 ls “|” sort

Las bibliotecas que uso son las mismas que las del ejercicio anterior.

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

    const int T = 4;

    if (argc != T){
        printf("Error en el numero de parametros.");
        exit(EXIT_FAILURE);
    }

    //redireccionar a la salida
    int fd[2];
    pid_t PID;

    if (strcmp(argv[2], "|")){
        pipe(fd); //llamada al sistema para crear un cauce sin nombre

        //comprobamos PID
        if(PID = fork() < 0){
            perror("\nError en el fork");
        }

        if (PID == 0){ //proceso hijo (redirecciona salida), es escritor
            close(fd[0]); //cierra el descriptor de lectura, ya que va a escribir.
            close(STDOUT_FILENO);

            if(fcntl(fd[1], F_DUPFD, STDOUT_FILENO) == -1){
                perror("\nFallo en el fcntl del hijo");
                exit(EXIT_FAILURE);
            }

            //ejecutamos la orden
            execlp(argv[1], argv[1], NULL);
        }

        else{ //proceso padre (redirecciona la entrada), es lector
            close(fd[1]); //cerramos descriptor de escritura, va a leer
            close(STDIN_FILENO);

            if(fcntl(fd[0], F_DUPFD, STDIN_FILENO) == -1){
                perror("\nError en fcntl del padre");
                exit(EXIT_FAILURE);
            }

            execlp(argv[3], argv[3], NULL);
        }
    }

    else{
        perror("El segundo parametro tiene que ser el indicado anteriormente\n");
        exit(EXIT_FAILURE);
    }

    return 0;
```

Programa 1:
El programa Programa 1 toma uno o varios pathnames como argumento y, para cada uno de los archivos especificados, se intenta un cerrojo consultivo del archivo completo. Si el bloqueo falla, el programa escanea el archivo para mostrar la información sobre los cerrojos existentes: el nombre del archivo, el PID del proceso que tiene bloqueada la región, el inicio y longitud de la región bloqueada, y si es exclusivo (“w”) o compartido (“r”). El programa itera hasta obtener el cerrojo, momento en el cual, se procesaría el archivo (esto se ha omitido) y, finalmente, se libera el cerrojo.

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

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

    struct flock cerrojo;
    int fd;

    //el programa toma uno o varios pathname como argumento
    //cuando ya no queden, se saldrá del bucle, ya que argc va disminuyendo.
    while (--argc > 0){
        //abre el pathname, y se puede tanto leer como escribir en él.
        if ((fd = open(*++argv, O_RDWR)) == -1) {
            perror("open fallo");
            continue;
        }

        //DEFINIMOS LOS CAMPOS DEL CERROJO, este cerrojo es consultivo,
        //se situa el cerrojo antes de realizar la operación E/S.
        cerrojo.l_type = F_WRLCK; //tipo de bloqueo --> cerrojo de escritura

        //estos son el rango de bytes que se quieren bloquear
        //En este caso se bloquea desde el inicio del archivo (SEEK_SET)
        //Hasta el final, porque se ha especificado 0 en l_start y l_len
        cerrojo.l_whence = SEEK_SET; //desde el inicio del archivo
        cerrojo.l_start = 0; //desde el inicio se bloquea hasta el finañ
        cerrojo.l_len = 0; //desde el inicio se bloquea hasta el final

        //Establece el cerrojo sobre el archivo fd con la orden F_SETLK
        /* intentamos un bloqueo de escritura del archivo completo */
        //Mientras de error, -1, vemos quien lo bloquea
        while (fcntl(fd, F_SETLK, &cerrojo) == -1){

            /*si el cerrojo falla, vemos quien lo bloquea*/
            //F_UNLCK elimina un cerrojo.
            //Comprobamos si el cerrojo falla, y con F_GETLK vemos quién lo bloquearía.
            //Si existe bloqueo, significa que el fcntl es != -1,entonces si existe
            //bloqueo y no se elimina el cerrojo, imprimimor la zona de bloqueo
            while(fcntl(fd, F_GETLK, &cerrojo) != -1 && cerrojo.l_type != F_UNLCK ){
                printf("%s bloqueado por %d desde %lld hasta %lld para %c", *argv,
                cerrojo.l_pid, cerrojo.l_start, cerrojo.l_len,
                cerrojo.l_type==F_WRLCK ? 'w':'r');

                if (!cerrojo.l_len) break;
                cerrojo.l_start +=cerrojo.l_len;
                cerrojo.l_len=0;
            } /*mientras existan cerrojos de otros procesos*/

        } /*mientras el bloqueo no tenga exito */

        /* Ahora el bloqueo tiene exito y podemos procesar el archivo */
        printf ("procesando el archivo\n");
        /* Una vez finalizado el trabajo, desbloqueamos el archivo entero */
        cerrojo.l_type =F_UNLCK; //se elimina el cerrojo
        cerrojo.l_whence =SEEK_SET;
        cerrojo.l_start =0;
        cerrojo.l_len =0;

        //con F_SETLKW establece un cerrojo, y  si este no se puede obtener, bloquea
        //a fcntl hasta que se pueda obtener dicho cerrojo.
        if (fcntl(fd, F_SETLKW, &cerrojo) == -1)
            perror("Desbloqueo");
    }
    return 0;
}
```

MAPEO

Ejemplo de creación de una proyección

```c
//EJEMPLO DE CREACIÓN DE UNA PROYECCIÓN//
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

//definimos constante de mmap
const int MMAPSIZE=1024;

int main(){

    int fd, num;
    char ch='\0';
    char *memoria;

    //abrimos el archivo tal que se pueda leer y escribir, y con O_EXCL onliga a que
    //el archivo se cree
    fd = open("Archivo", O_RDWR|O_CREAT|O_EXCL, S_IRWXU);

    //Si es -1 (error), significa que el archivo ya existe
    if (fd == -1) {
        perror("El archivo existe");
        exit(1);
    }

    //Va escribiendo en fd, de 1 en 1 el char \0, null,  con el número de bytes a
    // escribir sizeof(char), y así 1024 veces. Si la escritura da -1, significa que hay un
    //error en la escritura y por tanto no se puede escribir.
    for (int i=0; i < MMAPSIZE; i++){
        num = write(fd, &ch, sizeof(char));
        if (num!=1)
            printf("Error escritura\n");
    }

    //Creamos la proyección con mmap, una vez tenemos fd relleno.
    //address = 0, dirección de inicio (es lo que se pone normalmente)
    //size: el de MMAPSIZE,
    //FLAGS: los datos se pueden leer y escribir y además se complementa con que
    //las modificaciones serán compartidas
    //fd: el descriptor de archivo que se proyectará
    //offset: es el desplazamiento desde el cuál se proyecta, en este caso es 0
    memoria = (char *)mmap(0, MMAPSIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

    //si la memoria falla, no se podrá realizar la proyección
    if (memoria == MAP_FAILED) {
        perror("Fallo la proyeccion");
        exit(2);
    }

    close(fd); /* no es necesario el descriptor*/

    strcpy(memoria, "Hola Mundo\n"); /* copia la cadena en la proyección */

    exit(0);
}
```