Sistemas Operativos

Formulario de auto-evaluación

Modulo 2. Sesión 7. Construcción de un spool de impresión

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 1h 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:

Proxy

3. Comentarios y sugerencias:

b) Cuestionario de conocimientos adquiridos.

El código de mi programa **servidor.c** ha sido:

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

#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>




//Constantes
#define tamanio 1024
#define longitud 50

void proxy(int pid){

        char *n;
        int fd;

        //Genero el n
        sprintf(n, "/tmp/fifo.%d",pid);

        //Creo un FIFO
        mkfifo(n,0777);

        //Abro el FIFO
```

```
        fd=open(n,O_RDWR);
        if(fd<0){
                perror("Error en write\n");
                exit(EXIT_FAILURE);
        }

        //Redirigo la entrada estándar al archivo
        dup2(fd, STDIN_FILENO);

        //Ejecuto el proxy
        execlp("./proxy", "./proxy", NULL);

        //Si no pasa al codigo del proxy es que ha dado error, asi que lo pintamos
        perror("Fallo en execlp");
        exit(EXIT_FAILURE);

}

//Punto de entrada al programa
int main(int argc, char **argv){

        //Con esto redirigimos STDERR_FILENO --> STDOUT_FILENO
        dup2(STDOUT_FILENO, STDERR_FILENO);
        //dup2(1, 2);

        int buf[tamanio];
        int dfifos, dfifoe, nb;

        char nfifoe[longitud], nfifos[longitud];
```

```
setbuf(stdout, NULL);

if (argc != 2) {
        printf ("Uso: %s <n_fifo>\n",argv[0]);
        exit(1);
}

// Compone los ns de los FIFOs conocidos a partir del parametro,
// uno de entrada y otro de salida (desde el punto de vista del servidor).
sprintf(nfifoe,"%se",argv[1]);
sprintf(nfifos,"%ss",argv[1]);

//borramos los archivos fifo por si existieran previamente
unlink(nfifoe);
unlink(nfifos);

int pid;

umask(0);
//Creamos un cauce con n para la entrada
if((mkfifo(nfifoe, 0777)) == -1){
        perror("Error en mkfifo\n");
        exit(EXIT_FAILURE);
}

//Creamos un cauce con n para la salida
if((mkfifo(nfifos, 0777)) == -1){
        perror("Error en MKFIFO\n");
```

```
		exit(EXIT_FAILURE);
	}


		printf ("crear cauce de salida\n");



	//Abrimos los cauces
	dfifos=open(nfifos,O_RDWR);
	dfifoe=open(nfifoe,O_RDWR);

		printf ("cauces creados\n");



	nb=read(dfifos,buf,sizeof(buf));
	while(nb>0){

		printf ("Leyendo salida de FIFO\n");

		pid=fork();
		if(pid==0){
			proxy(getpid()); //Hijo

		} else if (pid > 0) {

			//Padre
			printf("padre pid es %d\n",pid);
			if(write(dfifoe,&pid,sizeof(int))<0){
				perror("Error en write\n");
				exit(EXIT_FAILURE);
				}
			} else if (pid == -1) {
```

```
                        perror("fallo en fork\n");
                        exit(EXIT_FAILURE);
                }

                nb=read(dfifos,buf,sizeof(buf));
        }

        if(nb==-1){
                perror("\nError en read\n");
                exit(EXIT_FAILURE);
        }

        pid = wait(NULL);
        while (pid > 0) {
                pid = wait(NULL);
        }

        /* si hay error, ignoramos si no hay más hijos a los que esperar. */
        if (pid == -1 && errno != ECHILD) {
                perror("fallo en wait");
                exit(EXIT_FAILURE);
        }

        exit(EXIT_SUCCESS);
}
```

El código de mi programa **proxy.c** ha sido:

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

//Constantes
#define tamanio 1024
#define longitud 50

void bloqueodesbloqueo(int dbloqueo, int orden) {

	struct flock cerrar;

	cerrar.l_type= orden;
	cerrar.l_whence= SEEK_SET;
	cerrar.l_start= 0;
	cerrar.l_len = 0;

	if (fcntl(dbloqueo, F_SETLKW, &cerrar) == -1) {
		perror ("Proxy: problemas al bloquear para la impresion");
		exit(1);
	}
}

//Punto de entrada al programa
int main(int argc, char **argv){
```

```
dup2(STDOUT_FILENO, STDERR_FILENO);

int buf[tamanio];
int nb;

FILE *tmpFile = tmpfile();
int orden=0;

//Leer datos de la entrada estándar
nb=read(STDIN_FILENO,buf,sizeof(buf));



//Comprobamos si hay algo en el FIFO
if(nb==-1){
        perror("Error en la lectura, en el proxy.\n");
        exit(EXIT_FAILURE);
}

//Si no da error se empieza a leer
while(nb>0){

        if(fwrite(buf,sizeof(char),nb,tmpFile)==-1){

                perror("Error en fwrite la escritura del proxy1\n");
                exit(EXIT_FAILURE);

        }

        nb=read(STDIN_FILENO,buf,sizeof(buf));
        if(nb==-1){
```

```
                perror("Error en la lectura, en el proxy2\n");
                exit(EXIT_FAILURE);

        }
}


if(nb==0){

        nb=fread(buf,sizeof(char),sizeof(buf),tmpFile);

        if(nb==-1){
                perror("Error en fread la lectura en del proxy.\n");
                exit(EXIT_FAILURE);
        }

        while(nb>0){

                //comento el desbloqueo un momento
                //bloqueodesbloqueo(STDOUT_FILENO, F_WRLCK);
                if(write(STDOUT_FILENO,buf,nb)==-1){
                        perror("Error en la escritura del proxy_1\n");
                        exit(EXIT_FAILURE);
                }

                nb=fread(buf,sizeof(char),sizeof(buf),tmpFile);
                if(nb ==- 1) {
                        perror("Error en la fread lectura en el proxy_2\n");
                        exit(EXIT_FAILURE);
                }
                orden++;
```

```
                }


                if(nb==0){
                        //bloqueodesbloqueo(STDOUT_FILENO, F_UNLCK);
                        fclose(tmpFile);
                        exit(EXIT_FAILURE);
                }

        }//if(nb==0)

}
```