Sistemas Operativos

Formulario de auto-evaluación

Modulo 2. Sesión 4. Comunicación entre procesos utilizando cauces

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 …… minutos.

1. He resuelto todas las dudas que tenía antes de iniciar la sesión de prácticas:si. 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 al **ejercicio 1** ha sido:

Para que ambos programas vayan sincronizados, he ejecutado el programa consumidorFIFO en segundo plano, de modo que quede bloqueado (con la llamada al sistema read) esperando a que el proceso productor escriba en ARCHIVO_FIFO para poder consumir o leer. Después he lanzado en primer plano el programa productorFIFO, con una cadena de caracteres para que escriba en ARCHIVO_FIFO y ese "mensaje" pueda see leido por el programa consumidor.

En caso de que el programa ConsumidorFIFO lea "fin", cerramos el cauce, de modo que el productor no pueda seguir escribiendo datos en el cauce (?)

Mi solución a la **ejercicio 2** ha sido:

En el siguiente programa se utiliza un cauce sin nombre, con dos procesos (padre e hijo).

En este caso el hijo cierra el descriptor fd[0], utilizado para lectura y abre el descriptor fd[1] u sado para escritura, pues el proceso hijo solo cumple la función de escribir en el cauce.

Dentro del else{} llevaría a cabo la labor de leer del cauce el proceso padre lo que ha escrito anteriormente el proceso hijo, en caso de que todavía no haya escrito nada quedaría bloqueado.

Cuando escribimos el mensaje terminamos con la llamada a exit(0), lo que supone el no volver a escribir en el cauce, y una vez consumido el dato, el programa acaba.

Mi solución a la **ejercicio 4** ha sido:

El programa tarea8.c hace lo mismo que tarea7.c solo que de una forma más reducida y eficaz, pues utilizando dup2 copiamos el descriptor de fichero en STDOUT_FILENO o STDIN_FILENO para leer o escribir en la entrada o salida estándar en vez de cerrar el cauce de STDOUT_FILENO (pantalla) para luego con dup duplicar el fd[1] que sería el cauce de escritura para que salga en pantalla (pues es la única casilla que tenemos libre de la tabla de archivos donde se puede alojar el cauce) y con el proceso padre pasaría mas de lo mismo: cerramos el descriptor de escritura, cerramos STDIN_FILENO (teclado) y luego duplicamos el cauce de lectura en el "teclado" pues es la primera casilla de la tabla de archivos abiertos que se ecuentra libre.

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

En primer lugar mostraré el programa esclavo que he realizado:

```
/*
programa esclavo para el ejercicio 5 de la sesión 4
Trabajo con llamadas al sistema del Subsistema de Procesos y Cauces conforme a POSIX 2.10
*/


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

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

	int extremo_anterior = atoi(argv[1]);
	int extremo_posterior = atoi(argv[2]);

	for(int i = extremo_anterior; i < extremo_posterior; i++){
		int d = 0;
		for(int k=1; k < i; k++){
			if(i%k == 0) // contabilizamos sus divisores y si son > 2 es porque no es
				d++;   //  primo

		}
		if(i != 1 && d < 2) // pues el 1 no puede ser nunca divisor
			printf("%d\n", i);
	}
```

```
	return(0);
}
```

En segundo lugar mostraré el programa maestro:

```
/*
Ana Alicia Vilchez Ceballos, ETSIIT(UGR)
tarea8.c
Programa del maestro esclavo llevando a cabo el uso de pipes y la redirección de entrada y
salida
*/

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

int main(int argc, char *argv[])
{
	int fd1[2]; // cauce de comunica ción entre el maestro y el primer esclavo
	int fd2[2]; // cauce de comunicación entre el maestro y el segundo esclavo
	pid_t PID;

	pipe(fd1); // Llamada al sistema para crear un pipe
	pipe(fd2); // Llamada al sistema para crear un pipe
	char cadena[10];

	char buffer[80]; // almacenamos los datos de la primer ejecucion del programa esclavo
	char buffer2[80]; // almacenamos los datos de la segunda ejecucion del programa esclavo
	int extremo_anterior = atoi(argv[1]);
```

```
int extremo_posterior = atoi(argv[2]);
// calculamos el intervalo que le deberemos pasar al programa esclavo
int mitad = (extremo_posterior - extremo_anterior)/2;

if ( (PID= fork())<0) { // creamos el primer proceso hijo
        perror("\Error en fork");
        exit(-1);
}

if (PID == 0) {
        //Cerrar el descriptor de lectura en cauce situado en el proceso hijo
        close(fd1[0]);

        //Duplicar el descriptor de escritura de cauce en el descriptor
        //correspondiente a la salida estandar, cerrado previamente en
        //la misma operación
        close(STDOUT_FILENO);
        dup(fd1[1]);

        sprintf(cadena, "%d", extremo_anterior+mitad);
        execlp("./esclavo","esclavo",argv[1], cadena, NULL);

}
else{
        if( (PID= fork())<0) { // creamos el segundo proceso hijo
                perror("\Error en fork");
                exit(-1);
        }
```

```
		if(PID == 0){
			//Cerrar el descriptor de lectura en cauce situado en el proceso hijo
			close(fd2[0]);

			//Duplicar el descriptor de escritura de cauce en el descriptor
			//correspondiente a la salida estandar, cerrado previamente en
			//la misma operación
			close(STDOUT_FILENO);
			dup(fd2[1]);

			sprintf(cadena, "%d", extremo_anterior+mitad+1);
			execlp("./esclavo","esclavo",cadena, argv[2], NULL);
			printf("Entra en el else if\n");

		}
		else{ // proceso padre---------->
			printf("Aqui se muestran los numeros primos del rango [%d, %d]:\n",extremo_anterior, extremo_posterior);
			//Cerrar el descriptor de ESCRITURA en el proceso padre
			close(fd1[1]);

			//Duplicar el descriptor de lectura en cauce en el descriptor
			//correspondiente a la salida estda r (stdout), cerrado previamente en
			//la misma operación
			close(STDIN_FILENO);
			dup(fd1[0]);

			// ahora leemos de fd1[0]
			while(read(fd1[0],&buffer, 80) > 0){
				printf("%s", buffer);
			}
```

```
                    //Cerrar el descriptor de ESCRITURA en el proceso padre
                    close(fd2[1]);

                    //Duplicar el descriptor de lectura en cauce en el descriptor
                    //correspondiente a la salida estda r (stdout), cerrado previamente en
                    //la misma operación
                    close(STDIN_FILENO);
                    dup(fd2[0]);

                    // ahora leemos de fd2[0]
                    while(read(fd2[0],&buffer2, 80) > 0){
                            printf("%s", buffer2);
                    }
            }
        }

        return(0);
}
```