Sistemas Operativos

Formulario de auto-evaluación

Modulo 2. Sesión 4. Comunicación entre procesos utilizando cauces

Nombre y apellidos:

Néstor Rodriguez Vico

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 ... 60... 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 al **ejercicio 1** ha sido:

Primero tenemos que ejecutar el consumidor en segundo plano y luego el productor pasándole como argumento el mensaje a escribir. El consumidor está en un bucle infinito e imprimirá el mensaje recibido hasta recibir la cadena “fin”.

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

Lo primero que hace este programa es crear un cauce sin nombre con la orden pipe. Después se crea un hijo con la ordena fork().

En el caso de que el código lo este ejecutando el hijo (PID == 0) se cierra el descriptor de lectura para si mismo. Después se envía el mensaje a través del cauce usando el descriptor de escritura.

En el caso de que el código lo este ejecutando el padre (PID != 0) cerramos el descriptor de escritura para si mismo y luego leemos los datos desde el descriptor de lectura. A continuación el padre imprime el numero de bytes que ocupa la cadena y la propia cadena.

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

El primer programa usa close para cerrar la salida estándar dejando la entrada del descriptor de lectura del hijo libre y luego usa dup para duplicar el descriptor de escritura en el cauce. Sucede llo mismo con el padre pero con la entrada estándar.

El segundo usa dup2, que hace lo mismo que las otras dos pero en una sola orden (cierra el descriptor antiguo y duplica el descriptor). La llamada es atómica. Se ha creado un cauce en el que el proceso hijo ejecuta la orden ls y lo redirecciona al descriptor de escritura de salida, con lo cual el proceso padre recibe la información de la orden que ejecutó el hijo y ejecuta sort.

En cuanto a ejecución se refiere no hay ninguna diferencia.

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

```
// esclavo.c


#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <math.h>
```

```
int Primo(int n){
        int i;
        int limite = sqrt(n);
        int primo = 1;
        for (i = 2; i <= limite && primo; i++)
                if (n % i == 0)
                        primo = 0;
        return primo;
};

int main(int argc, char *argv[]){
        int inicio, fin, i;
        inicio = atoi(argv[1]);
        fin = atoi(argv[2]);
        for (i = inicio; i < fin; i++)
                if (Primo(i))
                        write(STDOUT_FILENO, &i, sizeof(int));
        return 0;
}

// maestro.c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
   int fd1[2], fd2[2], leido, leido2, val1, val2;
```

```
int inicio, final, inicio2, final2, inicio3, final3;
pid_t esclavo1, esclavo2;
char aux[10], aux2[10];

if (argc < 3) {
    perror("Uso: ./maestro inicio fin.\n");
    exit(-1);
}

inicio = atoi(argv[1]);
final = atoi(argv[2]);
inicio2 = inicio;
final2 = ((final + inicio) / 2) - 1;
inicio3 = final2 + 1;
final3 = final;

pipe(fd1);
pipe(fd2);
printf("Primos entre %d y %d:\n", inicio, final);

esclavo1 = fork();
//Guardo en aux y en aux2 el intervalo del esclavo 1.
sprintf(axu, "%d", inicio2);
sprintf(aux2, "%d", final2);
if (esclavo1 == 0) {
    close(fd1[0]);
    dup2(fd1[1], STDOUT_FILENO);
    if (execl("./esclavo", "esclavo", aux, aux2, NULL) < 0) {
        perror("\nError al ejecutar ./esclavo");
        exit(-1);
    }
} else {
    close(fd1[1]);
```

```
        while ((leido = read(fd1[0], &val1, sizeof (int))) > 0) {
            printf("%d ", val1);
        }
        close(fd1[0]);
    }

    // Segundo esclavo
    esclavo2 = fork();
    sprintf(aux, "%d", inicio3);
    sprintf(aux2, "%d", final3);
    if (esclavo2 == 0) {
        close(fd2[0]);
        dup2(fd2[1], STDOUT_FILENO);
        if (execl("./esclavo", "esclavo", aux, aux2, NULL) < 0) {
            perror("\nError al ejecutar ./esclavo");
            exit(-1);
        }
    } else {
        close(fd2[1]);
        while ((leido2 = read(fd2[0], &val2, sizeof (int))) > 0) {
            printf("%d ", val2);
        }
        close(fd2[0]);
    }
    return 0;
}
```