Sistemas Operativos

Formulario de auto-evaluación

Modulo 2. Sesión 4. Comunicación entre procesos utilizando cauces

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 10 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:

Ninguna.

2. Tengo que trabajar algo más los conceptos sobre:

Los punteros a funciones, he tenido muchos problemas para manejar los datos de manera correcta. También he de revisar todo lo concerniente a la apertura y cierre de flujos.

3. Comentarios y sugerencias:

Ninguna.

b) Cuestionario de conocimientos adquiridos.

Mi solución al **ejercicio 1** ha sido:

El ejercicio propuesto consta de dos archivos, por un lado el productor y por otro el consumidor. El consumidor crea un archivo fifo para la comunicación, a partir su otra función es quedarse atrapado leyendo el archivo. Cuando encuentre la palabra fin cerrará el archivo y su proceso. En otro caso devuelve por pantalla la cadena que ha enviado el productorFIFO.

Por otro lado tenemos el productor, su función es abrir el archivo fifo y escribir en el contenido que le pasamos como argumento y, posteriormente se cierra.

La conclusión es que tenemos un productor que ha de ejecutarse en segundo plano y que a nuestros ojos quedará como en un bucle. Realemente está esperando porque el archivo fifo está vacío y no hay ninguna cadena a leer. Ejecutaremos individualmente el consumidor para enviar un mensaje a productor escribiendo en el archivo FIFO.

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

El programa actual implementa un proceso doble. Por un lado un proceso padre creará a un hijo. Este hijo será el encargado de escribir y el proceso padre de leer.

Para este caso en concreto usamos la instrucción fifo() que sobre un descriptor de archivos lo que hace es tener un vector de dos posiciones 0 y 1, donde la 0 se reserva para la lectura y 1 para la escritura. De este modo en el hijo lo primero que hacemos es cerrar sobre este descriptor la lectura, posteriormente y usando el descriptor de escritura escribiremos el primer mensaje. Por otro lado el padre lo que hace es cerrar el descriptor de la escritura y haciendonoos uso de un buff sobre el descriptor de lectura guardamos el tamaño de la cadena y la mostramos.

La instrucción pipe() crea un cauce sin nombre, toma un argumento de entrada que ha de ser un vector (int) con dos posiciones. En la primera como he explicado antes se usará para el flujo de entrada (lectura), es la posición 0 y salida o escritura que es la posición 1.

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

Ambos códigos hacen algo similar, ejecutar un ls | sort mostrandolo por la terminal gracias a haber redireccionado las salidas, el ls lo ejecuta el hijo y el sort el padre.

Por un lado tenemos dos funciones distintas dup(int old) y dup2(int old, int new). En tarea 7 lo que hacemos es usar dup() para, en el hijo primero cerrar la lectura del descriptor, despues el de la pantalla y hacemos dup(fd[1]) para que la salida se haga por el pipe.

Cuando el hijo acabe abrá ejecutado ls y este resultado se ‘quedará escrito en el pipe’, el padre recibirá esta información. Lo primero que hace es cerrar el descriptor de escritura, ya que vamos a leer, posteriormente cierra la salida standar (pantalla) y con dup([0]) se reedirige la entrada de modo que el teclado pasará a ser la información que está en el pipe, la que el hijo ha producido.

Por otro lado en la tarea8 usamos dup2() que lo que hace es simplificar el código anterior. El proceso que hacíamos de cerrar la salida/entrada del dup() desaparece ya que dup2() cierra automaticamente el descriptor viejo (STDOUT/IN_FILNE0, segundo argumento) y la sustituye por la salida/entrada del primer argumento.

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

```
/*
Programa esclavo
*/
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <math.h>
#include <sys/wait.h>

int es_primo(int N){
  int k, raiz;
  raiz = (int) sqrt(N);

  for( k=2; N%k && k<=raiz; k++);
    if(k==raiz+1)
      return 1;

  return 0;
}

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

  char buf_aux[1];

  if(argc!=3) {
    //Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
    printf("Modo de uso: %s <inicio> <fin>\n\n", argv[0]);
    exit(1);
  }

  int ini = atoi(argv[1]), fin = atoi(argv[2]);
  for (int i = ini; i <= fin; i++) {
    int escribo = es_primo(i);
    if (escribo){
      sprintf(buf_aux, "%i", i);
      write(STDIN_FILENO,buf_aux,sizeof(int));
    }
  }

  // Así sé cuando tengo que parar con el padre.
  sprintf(buf_aux, "%d", -1);
  write(STDIN_FILENO,buf_aux,sizeof(char));

  return (0);
}
```

```
/*
Programa maestro
*/

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

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

  // Compruebo los argumentos de entrada
  if(argc!=3) {
    //Si no se le han pasado los parámetros correctos muestra un mensaje de ayuda
    printf("Modo de uso: %s <inicio> <fin>\n\n", argv[0]);
    exit(1);
  }

  // Variables
  pid_t PID[2];
  int fd1[2], fd2[2];
  int bytes_leidos, inicio, final, medio;
  char param_1[10], param_2[10], param_3[10];
  char buf[10];

  // Calculamos las variables
  inicio = atoi(argv[1]);
  final= atoi(argv[2]);
  medio = inicio + (final - inicio) / 2;

  // Reconversiones
  sprintf(param_1,"%d", inicio);
  sprintf(param_2,"%d", final);
  sprintf(param_3,"%d", medio);

  //Creamos primer flujo
  pipe(fd1);

  // Creamos primer hijo:
  if ((PID[0] = fork())<0) {
    perror("Error al hacer fork");
    exit(-1);
  }

  if(PID[0]==0) { // Estoy en el hijo
    printf("Hijo 1 creado rango: [%s,%s]\n", param_1, param_3);
    close(fd1[0]);
    dup2(fd1[1],STDIN_FILENO);
    execlp("./esclavo", "esclavo", param_1, param_3,NULL);
    exit(0);
  }
```

```
  else{ // Vuelvo a estar en el padre
    printf("Esperando a hijo\n");
    wait(PID[0]); // Espero a mi primer hijo
    printf("Leyendo hijo\n");
    while((bytes_leidos = read(fd1[0], &buf, sizeof(int))) == sizeof(int)){
      printf("%s\n", buf);
      if(bytes_leidos < 0)
        printf("Error en leer primo [1]\n");
    }

    // Creo segundo flujo
    pipe(fd2);

    // Creo un nuevo hijo
    if((PID[1]=fork()) < 0){
      perror("Error al hacer fork");
      exit(-1);
    }

    if (PID[1] == 0) {  // Estoy en el hijo
      printf("Hijo 2 creado rango: [%s,%s]\n", param_3, param_2);
      close(fd2[0]);
      dup2(fd2[1],STDIN_FILENO);
      execlp("./esclavo", "esclavo", param_3, param_2,NULL);
      exit(0);
    }
    else{ // Soy el padre y vuelvo a leer
      while((bytes_leidos = read(fd2[0], &buf, sizeof(int))) == sizeof(int)){
        printf("%s\n", buf);
        if(bytes_leidos <0)
          printf("Error al leer algún primo [2]\n");
      }

    }
  }

  return(0);
}
```