Programación Avanzada

Curso 2023/2024

prof. Claudio Rossi

Practica 2: comunicación a través de "pipes"

Objetivos:

1. Uso de pipes

Comandos de S.O.: --

Las *pipes (*o "tuberías") son una primitiva de comunicación inter-proceso POSIX. A diferencia de otros métodos de comunicación entre procesos (IPC), una *pipe* sirve para comunicación en un solo sentido. Para comunicación de dos vías entre procesos, se pueden definir dos *pipes*, es decir, una para cada dirección. Las *pipes* se pueden usar únicamente para comunicación entre procesos que tienen una relación de parentesco, es decir, son padre e hijo o bien comparten un mismo padre.

Funciones utilizadas:

- int *pipe(int filedes[2])*: creación de una pipe. Los descriptores de los "extremos" de la pipe (salida y entrada) son almacenados en *filedes[0]* y *filedes[1]* respectivamente. Retorna 0 en caso de exito, -1 en caso de error.

- ssize_t *write(int filedes, const void *buffer, size_t size)*: escribe la cantidad *size* de bytes del *buffer* en el descriptor *fideles*. La función devuelve el numero de bytes escritos, o –1 en caso de fallo.

- ssize_t *read(int filedes, void *buffer, size_t size)*: lee la cantidad *size* de bytes del descriptor *filedes*, almacenándolos en *buffer*. Devuelve la cantidad de bytes efectivamente leídos, o -1 en caso de error.

- int *close(int filedes)*: cierra el descriptor. La función close devuelve un 0 en condiciones normales, o –1 en caso de fallo.

- int *sprintf(char *buffer, const char *formato,...)*: su funcionamiento es idéntico a la función *printf*, salvo que la salida generada se pone en el array apuntado por *buffer*. El valor devuelto es igual al número de caracteres puesto en el array.

- void *perror*(const char *message): imprime un mensaje de error asociado a la cadena que tiene como argumento.

- int *atoi(const char *cad)*: convierte la cadena de caracteres que apunta *cad* a un valor int. La cadena debe contener un número válido de entero.

Parte 1.
El proceso padre crea una pipe, y sucesivamente usa la función fork para crear un proceso hijo. Los dos se comunicarán a través de la *pipe*. Nótese que un proceso escribe en un extremo, y el otro lee desde el otro extremo. (Cada proceso cierra el extremo que no usa.)

Programa 1

```
/* Programa de ejemplo del uso de pipes */

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


int main(void)
{
  int fd[2];
  int status;
  int nread;
  char buffer[100];


  /* Creacion de la pipe */

  if(pipe(fd) == -1){
      perror("pipe");
      exit(EXIT_FAILURE);
      }

  printf("Pipe OK!\n");

  // Creacion de un proceso hijo
  switch(fork())
      {
      case -1:
        perror("fork");
        exit(EXIT_FAILURE);
      case 0:
        // este es el hijo ¡
        // Cierre del descriptor de escritura de la pipe en el
proceso hijo
        if(close(fd[1]) == -1)
            perror("close");


        // Lectura de la información contenida en la pipe.
        switch(nread = read(fd[0],buffer,sizeof(buffer)))
            {
            case -1:
              perror("read");
              break;
            case 0:
              perror("EOF");
              break;
            default:
```

```
              buffer[nread]='\0'; //Añade final de cadena de car. x printf
              printf("Hijo:    he    leido    %d    bytes    ('%s')
\n",nread,buffer);
              exit(EXIT_SUCCESS);
            }
      }

  // este es el padre

  /* Cierre del descriptor de lectura en el proceso padre */
  if(close(fd[0]) ==- 1)
      perror("close");

  /* El proceso padre escribe en la pipe */
  if(write(fd[1],"hola hijo",9) == -1)
      perror("write");

  /* El proceso padre espera la finalizacion del proceso hijo */
  wait(&status);
  exit(EXIT_SUCCESS);
}
```

Parte 2.
Basándose en el programa proporcionado, escribir otro para que se creen dos *pipes*, una para comunicación de padre a hijo, y otra de hijo a padre. El Padre deberá adivinar un numero elegido por el hijo (entre 0 y 256). Para ello, enviará sus intentos al hijo a través de la pipe. El hijo contestará con un mensaje para decir si el numero es mayor, menor o si ha acertado. En caso de acierto los proceso terminan.

(Para convertir un numero en cadena de caracteres usar la función *sprintf*, para convertir una cadena de caracteres en un numero usar la función *atoi.*)

Ejemplo:

```
char buffer[100];
char msg[100];
int i,x;

sprintf(msg,"%d",x);  // transforma el numero 10 en la cadena "10"
intento=atoi(buffer); // transforma la cadena en numero (ej: "123"->123)
```

Para generar un numero al azar:

```
srandom(getpid());   // inicializa la semilla para los numeros
x=(int)(256.0*random()/RAND_MAX);   // x será un numero entre 0 y 256
```

Nota. Para un uso avanzado de la pipe, se puede usar la instrucción *fcntl*

```
int fcntl(int fildes, int cmd, ...)
```

que proporciona distintas formas de modificar el comportamiento de la pipe, por ejemplo “read” non-bloqueante, etc.