Comunicación Entre Procesos (Interprocess Communication, IPC)

Pipes

Una tubería (pipe, cauce o '|') consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de buffer de datos entre elementos consecutivos.

1. Un pipe provee una interfaz entre dos procesos.
2. Es un descriptor especial que, en lugar de conectar a un archivo, conecta a otro proceso.
3. Un pipe provee un canal de comunicación **unidireccional** entre dos procesos que cooperan.

Pipes en Linux

a) Pipes a nivel Shell

Cuando en el shell de UNIX se escribe "linux> **grep 'perro' poema | wc –l**" lo que sucede es lo siguiente:

1. El shell construye un *pipe*, que es un par de "archivos inexistentes", que tienen la cualidad de que, lo que se escribe en uno se lee en el otro.
2. Después el shell crea dos procesos diferentes, uno con **grep** y otro con **wc**. Esos procesos son procesos completamente independientes y corren al mismo tiempo, aprovechando la multitarea del sistema operativo. El proceso con el grep tiene redireccionada su salida estándar hacia uno de los "archivos inexistentes" del pipe, y el otro "archivo inexistente" la toma como entrada estándar para el comando wc.

3. Esto quiere decir que grep y wc se ensamblan mágicamente al mismo tiempo.

Pipes en Linux

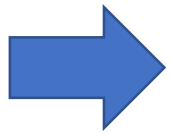
b) Pipes desde C

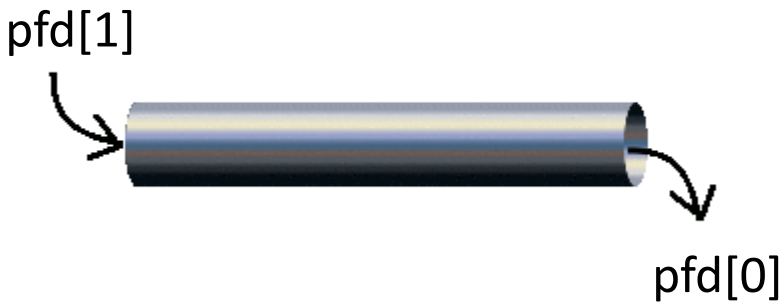
Para usar pipes desde C sólo hay que llamar a la función **pipe** y pedirle los dos descriptores de archivo, que serán los extremos del "tubo de comunicación“. Una tubería tiene dos descriptores de fichero: uno para el extremo de escritura y otro para el extremo de lectura. Como los descriptores de fichero de UNIX son simplemente enteros, un pipe o tubería no es más que un array de dos enteros, que se habrá de construir de la siguiente forma:

```c
#include <unistd.h>

int main()
{
    int pfd[2];

    /* Create a pipe.*/
    if (pipe(pfd) < 0) {
        perror("pipe");
        exit(1);
    }
    return 0;
}
```





Después de la invocación a **pipe** el arreglo de enteros `pfd` contiene en `pfd[0]` un descriptor de archivo para *leer*, y en `pfd[1]` un descriptor de archivo para *escribir*. Esto quiere decir que los pipes funcionan en una sola dirección.

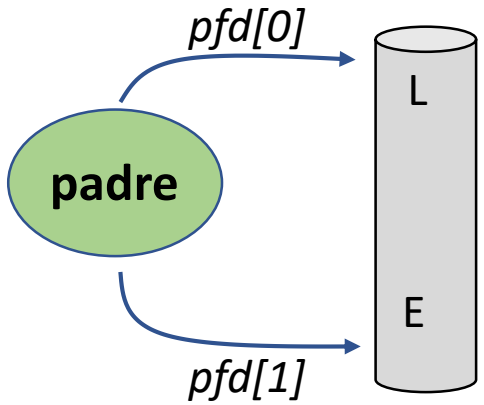
Pipes desde C

- Para crear la tubería se emplea la función pipe(), que abre dos descriptores de fichero y almacena su valor en los dos enteros que contiene el array de descriptores de fichero. El primer descriptor de fichero es abierto como **O_RDONLY**, es decir, sólo puede ser empleado para lecturas. El segundo se abre como **O_WRONLY**, limitando su uso a la escritura. De esta manera se asegura que el pipe sea de un solo sentido: por un extremo se escribe y por el otro se lee, pero nunca al revés.
- Una vez creado un pipe, se podrán hacer lecturas y escrituras de manera normal, como si se tratase de cualquier fichero.
- un proceso hijo hereda todos los descriptores de ficheros abiertos de su padre, por lo que la comunicación entre el proceso padre y el proceso hijo es bastante cómoda mediante una tubería.
- El proceso padre y su hijo comparten datos mediante una tubería.
- La tubería “pfd” se hereda al hacer el fork() que da lugar al proceso hijo, pero es necesario que el padre haga un close() de pfd[0] (el lado de lectura de la tubería), y el hijo haga un close() de pfd[1] (el lado de escritura de la tubería). Una vez hecho esto, los dos procesos pueden emplear la tubería para comunicarse (siempre unidireccionalmente), haciendo **write**() en pfd[1] y **read**() en pfd[0], respectivamente.

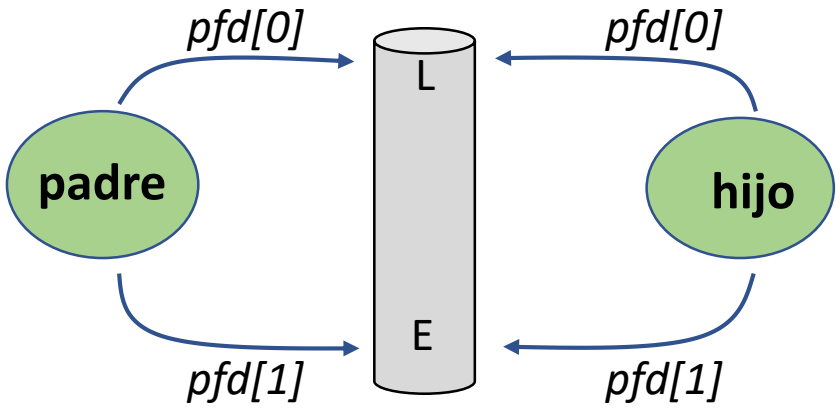
Funcionamiento *pipe* y forma de comunicación entre procesos

1. Cuando se ejecuta el programa se crea un proceso, para el caso, el proceso PADRE.

2. El proceso PADRE, crea una pipe (instrucción *pipe(pfd)* donde *pfd* es un arreglo de dos descriptores, ***pfd[0]*** apunta hacia el extremo de **lectura** de la pipe y ***pfd[1]*** al extremo de **escritura** de la pipe).

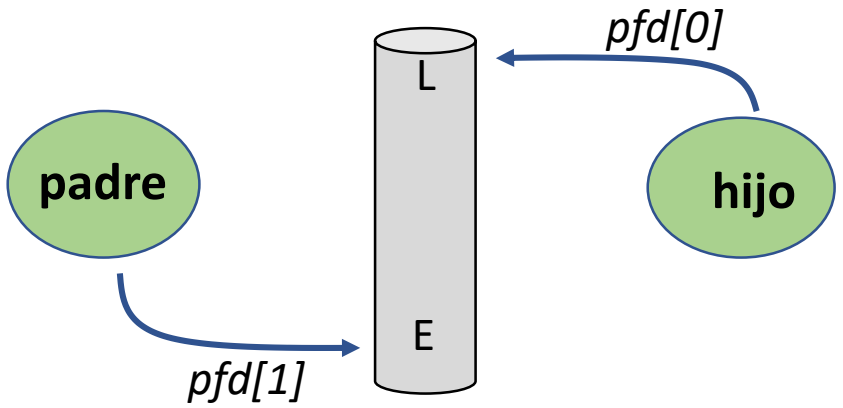


3. Después el PADRE debe crear al proceso HIJO (instrucción *fork*). Cuando el hijo es creado "**hereda**" los mismos descriptores del padre:



Funcionamiento con *pipe* y forma de comunicación entre procesos

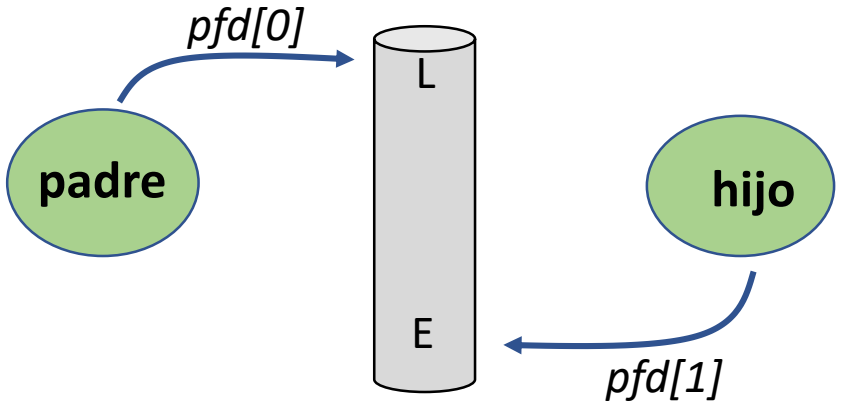
4. Si el PADRE quiere enviar datos al HIJO debe cerrar su descriptor *pfd[0]* y el HIJO cerrar su descriptor *pfd[1]*.



close(pdf[0]);

close(pdf[1]);

5. Si el HIJO quiere enviar datos al PADRE debe cerrar su descriptor *pfd[0]* y el PADRE cerrar su descriptor *pfd[1]*.



close(pdf[1]);

close(pdf[0]);