Práctica 3: E/S Multiplexada y Pipes

Ejercicio 1 Tuberías sin nombre

Escribir un programa que emule el comportamiento de la shell en la ejecución de una sentencia en la forma: `comando1 argumento 1 | comando2 argumento2`. El programa recibirá *comando1*, *argumento1*, *comando2* y *argumento2* como argumentos de las línea de comandos, es decir, para emular la ejecución de `ls -l | wc -l`, llamaremos a nuestro programa con los siguientes argumentos:

```
./ejercicio1 ls -l wc -l
```

Para emular este comportamiento el programa creará una tubería sin nombre (`int pipe(int pipefd[2])`) y a continuación creará un proceso hijo:

- En el proceso padre se ejecutará "*comando1 argumento1*", redireccionando previamente la salida estándar al extremo de escritura del pipe.
- En el proceso hijo se ejecutará "*comando2 argumento2*", redireccionandose previamente en este caso la entrada estándar al extremo de lectura del pipe.

Recordad que antes de ejecutar el comando correspondiente deben cerrarse todos los descriptores no necesarios tanto en el proceso padre como en el proceso hijo.

Ejercicio 2. Comunicación con Tuberías con Nombre

Las tuberías con nombre son un mecanismo de comunicación FIFO, útil para procesos sin relación de parentesco. La gestión de las tuberías con nombre es igual a la de un fichero ordinario (write, read, open...). A diferencia de los ficheros regulares, las tubería debe abrirse en los dos extremos (lectura/escritura), en caso contrario se bloquean las operaciones.

Ejercicio 2.1.

Usar el comando `mkfifo` para crear una tubería (ej. `\~/tuberia1`). Usar las herramientas del sistema de ficheros (*stat*, *ls*...) para determinar sus propiedades. Comprobar su funcionamiento usando utilidades para escribir y leer de ficheros (ej. *echo*, *cat*, *less*, *tail*).

Ejercicio 2.2.

Escribir un programa que abra la tubería con nombre anterior (`\~/tuberia1`) en modo solo escritura, y escriba en ella el primer argumento del programa. En otro terminal, leer de la tubería usando un comando adecuado.

Ejercicio 2.3.

Es habitual que un proceso lea o escriba de diferentes flujos. Las llamadas `select()`, `poll()` y `epoll()` permiten multiplexar diferentes las operaciones de E/S sobre múltiples flujos.

Escribir un programa que cree una nueva tubería con nombre (ej. `\~/tuberia2`) y espere con `select()` hasta que haya datos listos para leer en alguna de las dos tuberías. El programa debe leer entonces de la tubería correspondiente y mostrar los datos leídos en la salida estándar, indicando la tubería desde la que se leyó. Usar lecturas no bloqueantes usando el flag *O_NONBLOCK*.

Ejercicio 2.4

Completar el program del apartado anterior 2.3 añadiendo la posibilidad de que el programa termine cuando el usuario presione en el terminal `CTR+D`

Nota: El primer conjunto de descriptores de fichero que se le pasa a `select` es aquel sobre el que queremos esperar hasta que haya algún dato que leer. Hay una pequeña excepción. Cuando se alcanza el final de un fichero (o se corta la comunicación en una tubería o un socket), `select` se desbloquea si el objeto está entre los especificados en su primer conjunto de descriptores de fichero. En estos casos al hacer un `read` sobre uno de ellos, la llamada al sistema devolverá cero inmediatamente, señal que nos sirve para detectar el fin del fichero o el corte de la comunicación.

Ejercicio 3. Comunicación Bidireccional

Para la comunicación bi-direccional o full-duplex con pipes es necesario crear dos tuberías, una para cada sentido: padre_hijo e hijo_padre. Escribir un programa que implemente el siguiente protocolo de comunicación:

- El proceso padre leerá de la entrada estándar un mensaje (una cadena) y lo enviará al proceso hijo escribiendo en la tubería **padre_hijo**. Entonces permanecerá bloqueado esperando la confirmación por parte del hijo en la otra tubería, **hijo_padre**.
- El proceso hijo leerá de la tubería **padre_hijo**. Cuando reciba el mensaje del padre por la tubería:
  - Convertirá el mensaje a mayúsculas (consultar la función de la librería de C `toupper()`)
  - Lo mostrará por la salida estándar.
  - Esperará durante 1 segundo
  - Trascurrida la espera enviará el carácter ‘n’ al proceso padre por la tubería **hijo_padre** para indicar que está listo para recibir un nuevo mensaje. Cuando se hayan recibido 10 mensajes, se enviará el carácter ‘q’ al padre para indicarle que debe finalizar.