Práctica 2.4. Señales y Tuberías

Objetivos

En esta práctica veremos los comandos e interfaz para la gestión de señales y tuberías. Las señales informan de la ocurrencia de un evento de forma asíncrona. Las tuberías ofrecen un mecanismo sencillo y efectivo para la comunicación entre procesos en un mismo sistema.

Contenidos

Preparación del entorno

Esta práctica únicamente requiere las herramientas y entorno de desarrollo de usuario.

Señales

Ejercicio 1. `kill(1)` permite enviar señales a un proceso o grupo de procesos por su identificador (p`kill(1)` permite hacerlo por nombre de proceso). Estudiar la página de manual y las señales que se pueden enviar a un proceso.

Ejercicio 2. En un terminal, arrancar un proceso de larga duración (ej. `sleep 600`). En otra terminal, enviar diferentes señales al proceso, comprobar el comportamiento. Observar el código de salida del proceso. ¿Qué relación hay con la señal enviada?

Ejercicio 3. Escribir un programa que bloquee las señales `SIGINT` y `SIGTSTP`. Después de bloquearlas el programa debe suspender su ejecución con `sleep(3)` un número de segundos que se obtendrán de la variable de entorno `SLEEP_SECS`. Al despertar, el proceso debe informar de si recibió la señal `SIGINT` y/o `SIGTSTP`. En este último caso, debe desbloquearla con lo que el proceso se detendrá y podrá ser reanudado en la *shell* (imprimir una cadena antes de finalizar el programa para comprobar este comportamiento).

Ejercicio 4. Escribir un programa que instale un manejador para las señales `SIGINT` y `SIGTSTP`. El manejador debe contar las veces que ha recibido cada señal. El programa principal permanecerá en un bucle que se detendrá cuando se hayan recibido 10 señales. El número de señales de cada tipo se mostrará al finalizar el programa.

Ejercicio 5. Escribir un programa que realice el borrado programado del propio ejecutable. El programa tendrá como argumento el número de segundos que esperará antes de borrar el fichero. Para notificar el final de la espera el programa instalará una alarma con `alarm(2)`. El borrado del fichero se podrá detener si se recibe la señal `SIGUSR1` antes de que termine el periodo de espera.

Nota: Usar `sigsuspend(2)` para suspender el proceso y la llamada al sistema apropiada para borrar el fichero.

Tuberías sin nombre

Las tuberías sin nombre son entidades gestionadas directamente por el núcleo del sistema y son un mecanismo de comunicación unidireccional eficiente para procesos relacionados (padre-hijo). La forma de compartir los identificadores de la tubería es por herencia (con `fork(2)`).

Ejercicio 6. Escribir un programa que emule el comportamiento de la *shell* en la ejecución de una sentencia en la forma: `comando1 argumento1 | comando2 argumento2`. El programa creará una tubería sin nombre y creará un hijo:

- El proceso padre redireccionará la salida estándar al extremo de escritura de la tubería y ejecutará `comando1 argumento1`.
- El proceso hijo redireccionará la entrada estándar al extremo de lectura de la tubería y ejecutará `comando2 argumento2`.

Probar el funcionamiento con una sentencia similar a: `./pipe echo 12345 wc -c`

Nota: Antes de ejecutar el comando correspondiente, deben cerrarse todos los descriptores no necesarios.

Ejercicio 7. Para la comunicación bi-direccional, es necesario crear dos tuberías, una para cada sentido: `p_h` y `h_p`. Escribir un programa que implemente el mecanismo de sincronización de parada y espera:

- El padre leerá de la entrada estándar (terminal) y enviará el mensaje al proceso hijo, escribiéndolo en la tubería `p_h`. Entonces permanecerá bloqueado esperando la confirmación por parte del hijo en la otra tubería, `h_p`.
- El hijo leerá de la tubería `p_h`, escribirá el mensaje por la salida estándar y esperará 1 segundo. Entonces, enviará el carácter `'1'` al proceso padre, escribiéndolo en la tubería `h_p`, para indicar que está listo. Después de 10 mensajes enviará el carácter `'q'` para indicar al padre que finalice.

Tuberías con nombre

Las tuberías con nombre son un mecanismo de comunicación unidireccional, con acceso de tipo FIFO, útil para procesos sin relación de parentesco. La gestión de las tuberías con nombre es igual a la de un archivo ordinario. Revisar la información en `fifo(7)`.

Ejercicio 8. Usar `mkfifo(1)` para crear una tubería con nombre. Usar las herramientas del sistema de ficheros (`stat(1)`, `ls(1)`…) para determinar sus propiedades. Comprobar su funcionamiento usando utilidades para escribir y leer de ficheros (ej. `echo(1)`, `cat(1)`, `tee(1)`...).

Ejercicio 9. Escribir un programa que abra la tubería con el nombre anterior en modo sólo escritura, y escriba en ella el primer argumento del programa. En otro terminal, leer de la tubería usando un comando adecuado.

Multiplexación síncrona de entrada/salida

Es habitual que un proceso lea o escriba de diferentes canales de E/S. `select(2)` permite multiplexar las diferentes operaciones de E/S sobre múltiples canales.

Ejercicio 10. Crear otra tubería con nombre. Escribir un programa que espere hasta que haya datos listos para leer en alguna de ellas. El programa debe mostrar la tubería desde la que leyó y los datos leídos. Consideraciones:

- Para optimizar las operaciones de lectura usar un *buffer* (ej. de 256 bytes).
- Usar `read(2)` para leer de la tubería y gestionar adecuadamente la longitud de los datos leídos.
- Normalmente, la apertura de la tubería para lectura se bloqueará hasta que se abra para escritura (ej. con `echo 1 > tuberia`). Para evitarlo, usar la opción `O_NONBLOCK` en `open(2)`.
- Cuando el escritor termina y cierra la tubería, `read(2)` devolverá `0`, indicando el fin de fichero, por lo que hay que cerrar la tubería y volver a abrirla. Si no, `select(2)` considerará el descriptor siempre listo para lectura y no se bloqueará.