Introducción a Sistemas Operativos: Comunicación entre Procesos

Clips xxx
Francisco J Ballesteros

1. Terminales y sesiones

El *terminal* es el dispositivo en UNIX que representa la consola, ventana o acceso remoto al sistema utilizado para acceder al sistema. Ya hemos visto que `/dev/tty` representa el terminal en cada proceso y que hay otros dispositivos habitualmente llamados `/dev/tty*` o `/dev/pty*` que representan terminales concretos.

Nos interesa ser conscientes de este dispositivo puesto que es capaz de enviar señales a los procesos y además suministra la entrada/salida en la mayoría de programas interactivos. La idea es que el terminal controla los procesos que lo utilizan y, por ejemplo, es capaz de enviar la señal `INT` si se pulsa *control-c* y es capaz de enviar la señal `HUP` si el usuario sale del terminal (cierra la ventana, desconecta la conexión de red que ha utilizado para conectarse, lo hace un log-out de la consola).

La abstracción es sencilla: UNIX agrupa los procesos en **grupos de procesos** y cada uno pertenece a una **sesión**. A su vez, cada sesión tiene un **terminal de control** (en realidad cada proceso lo tiene).

Por ejemplo, si abrimos una ventana con un shell obtenemos un nuevo terminal. Podemos ver qué terminal estamos utilizando con el comando *tty(1)*. Por ejemplo, en una ventana tenemos

```
unix$ tty
ttys002
unix$
```

y en otra

```
unix$ tty
ttys007
unix$
```

tenemos un terminal distinto.

Pues bien, la *sesión* es la abstracción que representa la sesión en el sistema en cada uno de estos terminales. En nuestro ejemplo (y si no hay ninguna otra consola ni conexión remota, lo cual no es cierto) tendríamos dos sesiones.

Dentro de una sesión tenemos ejecutando un shell y podemos ejecutar una línea de comandos tal como

```
unix$ ls | wc -l
```

o cualquier otra. En este caso, tanto `ls` como `wc` terminarán perteneciendo al mismo *grupo* de procesos. Y si ejecutamos

```
unix$ (sleep 3600 ; echo hi there) &
unix$ ls | wc -l
```

tendremos dos grupos de procesos. La intuición es que podemos utilizar los grupos para agrupar los procesos (de una línea de comandos, por ejemplo).

Así pues tenemos procesos agrupados en grupos que están agrupados a su vez en sesiones. Cada una de estas abstracciones es simplemente otro tipo de datos más implementado por el kernel y como todo lo demás tendrá su propio record con los atributos que deba tener dentro del kernel. Una vez más es tan sencillo como eso.

Un proceso puede iniciar una nueva sesión llamando a

```
setsid();
```

lo que a su vez hace que también cree un nuevo grupo de procesos y se convierta en su líder. La función *setpgid(2)* es la que se utiliza para hacer que un proceso cree un nuevo grupo de procesos y pase a ser su líder (aunque seguirá dentro de la misma sesión).

Es poco habitual que tengas que llamar a estas funciones. Podrías desear que tu programa se deshaga del terminal de control para que continue ejecutando como un proceso en background sin que ningún terminal le envíe señal alguna (para que ejecute como un **demonio**, que es como se conoce a estos procesos). Esto sucederá si estás programando un servidor o cualquier otro programa que se desea que continue su ejecución independientemente de la sesión.

Pero incluso en este caso, lo normal es llamar a *daemon(3)* que crea un proceso utilizando *fork(2)* y hace que su directorio actual sea "`/`" y que no tenga terminal de control. De ese modo puede ejecutar sin molestar. Eso si, si dicho programa vuelve a utilizar un terminal para leer o escribir, seguramente adquiera dicho terminal como terminal de control. Aunque los detalles exactos dependen del tipo de UNIX concreto que utilices. Consulta tu manual.

El comportamiento del terminal está descrito en *tty(4)* y puede cambiarse. Desde el shell disponemos del comando *stty(1)* (*set tty*) y desde C disponemos de una llamada al sistema *ioctl(2)* que es en realidad una forma de efectuar múltiples llamadas que no tienen llamada al sistema propia (simplemente se utiliza una constante para indicar qué operación de control de Entrada/Salida queremos hacer y otros argumentos empaquetan los parámetros/resultados de la llamada en un record).

Por ejemplo, estos son los ajustes de nuestro terminal

```
unix$ stty -a
speed 9600 baud; rows 24; columns 58; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = M-^?; eol2 = M-^?; swtch = <undef>; start = ^Q;
stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr
icrnl ixon -ixoff -iuclc -ixany imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0
cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase
-tostop -echoprt echoctl echoke
unix$
```

Conviene no intimidarse por la cantidad de ajustes. Hemos mostrado todos utilizando el flag `-a` de `stty`. Los terminales eran artefactos mas o menos simples cuando UNIX se hizo allá por los 70, pero hoy día acarrean toda una historia de dispositivos pintorescos.

Una cosa que vemos es que la opción `intr` tiene como valor `^C`. Esto quiere decir que si pulsamos *control-c* el terminal envía la señal *INT* al grupo de procesos que está ejecutando en **primer plano** o **foreground**. Dicho de otro modo, a los involucrados en la línea de comandos que estamos ejecutando en el

momento de pulsar `^C`.

Aquellos procesos que hemos ejecutado el **background** o en **segundo plano**, esto es, aquellos que hemos ejecutado con un "&" en su línea de comandos pertenecen a otro grupo de procesos y no reciben la señal `INT` cuando pulsamos `^C`.

Podemos cambiar la combinación de teclas que produce la señal de interrumpir el grupo en primer plano. Esto podemos hacerlo por ejemplo como en

```
unix$ stty intr ^h
unix$
```

Aquí hemos escrito literalmente "`^h`", no hemos pulsado *control-h*. Tras la ejecución de este comando, *control-c* no interrumpe la ejecución de ningún comando en este terminal. En cambio, *control-h* sí que puede utilizarse para interrumpir al grupo que está ejecutando en foreground.

Para restaurar los valores por defecto o dejarlos en un estado razonable podemos ejecutar

```
unix$ stty sane
```

y volvemos a utilizar `^C` para interrumpir.

Otro ajuste interesante es el flag de `echo`. Si ejecutamos

```
unix$ ssty -echo
```

(quitar el eco) veremos que en las siguientes líneas de comandos no vemos nada de lo que escribimos. No obstante, si escribimos el nombre de un comando y pulsamos *enter* vemos que UNIX lo ejecuta normalmente.

```
unix$ stty -echo
unix$ Sat Aug 27 11:23:24 CEST 2016
```

El programa que escribe en pantalla lo que nosotros escribimos en el teclado es el terminal, y claro, si le pedimos que no haga eco de lo que escribimos, deja de hacerlo. Podría ser útil para leer contraseñas y para alguna otra cosa...

Para dejar el terminal en su estado normal podemos ejecutar

```
unix$ stty echo
```

aunque no veamos por el momento lo que escribimos. Aunque si es una ventana, es más simple cerrarla y abrir otra.

1.1. Modo crudo y cocinado

Un ajuste importante es el modo de **disciplina de línea** del terminal. Se trata de un parámetro que puede tener como valor *modo crudo* (o *raw*) o bien *modo cocinado* (*cooked*).

Como ya sabes el terminal procesa caracteres según los escribes pero no suministra esos caracteres a ningún programa que lea del terminal hasta que pulsas *enter*. A esto se le llama *modo cocinado*. El terminal "cocina" la línea para permitir que puedas editarla.

Pero en algunas ocasiones querrás procesar directamente los caracteres que escribe el usuario (por ejemplo, si estás implementando un juego o un editor y quieres atender cada carácter por separado). El *modo crudo* sirve justo para esto. Si lo activas, el terminal no cocina nada y se limita a darte los caracteres según estén disponibles. Naturalmente, no podrás borrar y, de hecho, el carácter que utilizabas para borrar pasará a comportarse como cualquier otro carácter.

Con que sepas que estos modos existen tenemos suficiente. Puedes utilizar el manual y las llamadas que hemos mencionado recientemente para aprender a utilizar estos modos, aunque es muy poco probable que lo necesites.