Introducción a Sistemas Operativos: Ficheros

Clips xxx
Francisco J Ballesteros

1. Enlaces simbólicos

Recientemente hemos visto que podemos enlazar ficheros desde varios nombres. A este tipo de enlaces se los denomina **hard links** o *enlaces duros*. La ventaja que tienen es que todos los enlaces se comportan bien. Por ejemplo, borrando cualquiera de los nombres para un fichero seguimos teniendo el fichero accesible.

No obstante, no es posible hacer un enlace a un fichero que esté en otro sistema de ficheros (en otra partición, otro disco y otra máquina). Esto es obvio si piensas que el enlace simplemente crea una entrada de directorio que se refiere a un número de *i-nodo* dado, y piensas que el número de *i-nodo* sólo tiene sentido dentro de la misma partición o sistema de ficheros.

Para solucionar este problema, UNIX dispone de **enlaces simbólicos** además de enlaces duros. Un enlace simbólico es similar a un *acceso directo* en Windows. Se trata de un fichero cuyos datos son el path de otro fichero. ¡Tan sencillo como eso!. La consecuencia es que un enlace simbólico puede apuntar a cualquier fichero dado su path. Y, naturalmente, la desventaja es que si borramos el fichero original perdemos los datos del fichero y el enlace queda apuntando a un fichero que no existe.

Para crear un enlace simbólico puedes utilizar el flag `-s` de *ln(1)*. Por ejemplo:

```
unix$ echo hola >fich
unix$ ln -s fich link
unix$ ls -l fich link
-rw-r--r--  1 nemo  wheel  5 Aug 21 19:24 fich
lrwxr-xr-x  1 nemo  wheel  4 Aug 21 19:24 link -> fich
unix$ cat link
hola
unix$ cat fich
hola
unix$ rm fich
unix$ cat link
cat: link: No such file or directory
unix$ ls -l link
lrwxr-xr-x  1 nemo  wheel  4 Aug 21 19:24 link -> fich
```

Es interesante fijarse en la salida de `ls` para `link`. El primer carácter es "`l`", lo que indica que el tipo de fichero es *symbolic link*. Si recuerdas *stat(2)*, puedes utilizar código como

```
if ((st.st_mode & S_IFMT) == S_IFLNK) {
    // el fichero es un enlace simbolico
}
```

para ver si el fichero que tienes entre manos es un enlace simbólico o no.

En general, las llamadas al sistema y funciones que operan con ficheros suelen seguir los enlaces como cabe esperar. Por ejemplo, si `open` recibe como argumento el path de un fichero que es un enlace

simbólico, UNIX se da cuenta de ello y procede como sigue:

- Se leen los datos del fichero
- Se interpretan como el path del que hay que hacer el `open`
- Se efectúa el `open` de dicho fichero

Esto hace que normalmente tus programas trabajen correctamente aunque encuentren enlaces simbólicos.

No obstante, hay ocasiones en que hay que prestar atención a este tipo de ficheros. Por ejemplo, si hacemos un `unlink` de un enlace simbólico tan sólo se borra el enlace y no el fichero enlazado. Aquí UNIX no sigue el enlace de forma automática. Pero es mejor así, dado que lo que cabría esperar al ejecutar

```
unix$ rm fich
```

es que se borre `fich`, sea un enlace o no. En ningún caso esperamos que se borre el fichero al que apunta `fich` si es un enlace simbólico.

Llamadas como *stat(2)* son más delicadas. Utilizar `stat` sobre un fichero que es un enlace simbólico hace que UNIX (como hace en general) atraviese el enlace automáticamente. Esto hace que en realidad obtengamos los metadatos del fichero al que apunta el enlace. Dicho de otro modo, ¡`stat` nunca te dirá que un fichero es un enlace simbólico!

Una forma de solucionar este problema es llamar a `lstat`, que funciona igual que `stat` pero cuando el fichero es un enlace devuelve los atributos del enlace en lugar de los del fichero enlazado.

Sucede lo mismo con llamadas como *chown(2)*, *chmod(2)*, etc. Estas llamadas atraviesan los enlaces simbólicos y operan sobre los ficheros enlazados, pero dispones de llamadas con igual nombre pero comenzando por "`l`" (por ej., "`lchmod`") que, cuando el fichero es un enlace simbólico, trabajan sobre el enlace en sí y no sobre el fichero enlazado.

¡El manual es tu amigo! Aunque creas que sabes usar UNIX, consulta rápidamente la página de manual de la llamada que piensas usar. Quizá recuerdes que debieras utilizar `lstat` y no `stat` o `fstat`... ¡y ahorrarás mucho tiempo depurando bugs que podrías haber evitado!

Para crear un enlace simbólico desde C puedes utilizar *symlink(2)* que funciona igual que *link(2)*, pero crea un enlace simbólico en lugar de uno duro. Por ejemplo, este programa es similar a *ln(1)* bajo el flag `-s`:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <err.h>

int
main(int argc, char* argv[])
{

    if (argc != 3) {
        fprintf(stderr, "usage: %s old new\n", argv[0]);
        exit(1);
    }
    if (symlink(argv[1], argv[2]) < 0) {
        err(1, "symlink %s", argv[1]);
    }
    exit(0);
}
```

2. Dispositivos

Ya conocemos diversos tipos de fichero:

- Ficheros regulares
- Directorios
- Enlaces simbólicos

Pero es hora de mencionar dos más:

- Dispositivos de tipo carácter
- Dispositivos de tipo bloque

Los dispositivos son *i-nodos* que no contienen datos en realidad. Se utilizan para representar dispositivos que pueden ser artefactos hardware (como una impresora o un disco) o pueden ser invenciones de software (como `/dev/null`). La mayoría de los dispositivos suelen estar en `/dev`.

UNIX distingue entre dispositivos de modo carácter y de modo bloque principalmente por que los primeros se espera que correspondan a streams de caracteres (el teclado, la pantalla al escribir caracteres, el ratón, etc.) y los segundos se espera que correspondan a almacenes de bloques de bytes de los que UNIX podría mantener una cache.

De hecho, los discos duros (y las particiones) suelen tener un par de ficheros de dispositivo cada uno: uno de tipo carácter y uno de tipo bloque. El primero se utiliza para formatear el disco y para leerlo saltándose la cache. El segundo se utiliza para acceder al disco beneficiándose de la cache de bloques que mantiene UNIX. Por ejemplo, presentamos los dispositivos de la primera partición del primer disco en nuestro sistema:

```
unix$ ls -l /dev/rdisk0s1
crw-r-----  1 root  operator    1,   1 Jul 13 07:30 /dev/rdisk0s1
unix$ ls -l /dev/disk0s1
brw-r-----  1 root  operator    1,   1 Jul 13 07:30 /dev/disk0s1
```

Cuando abres, lees o escribes un fichero, UNIX localiza el *i-nodo* del fichero y en función del tipo de fichero hace una cosa u otra. En el caso de los dispositivos, simplemente se llama a una función distinta (en el kernel) para cada tipo de dispositivo y se deja que ella haga el trabajo.

El código de un dispositivo suele llamarse **manejador** o **driver**. Si se trata de hardware, además de atender las interrupciones de dicho trozo de hardware y de detectar si está instalado en el sistema o no, el manejador implementará las operaciones del *i-nodo* para el tipo de dispositivo de que se trate.

En un *i-nodo* de un dispositivo tienes dos números:

- Número principal (*major number*)
- Número secundario (*minor number*)

El primero identifica un manejador de dispositivo (por ejemplo, discos duros SATA). El segundo identifica de qué dispositivo concreto se trata (por ejemplo, el primer disco SATA en el primer bus SATA).

Así pues, si un *i-nodo* es un dispositivo de bloque que identifica un disco y el número principal corresponde al driver de discos SATA, todas las llamadas `open`, `read`, ... que operen sobre ese *i-nodo* llamarán a funciones concretas del manejador de discos SATA. A dichas funciones se les dará además el *i-nodo* de que se trate y podrán ver el número secundario (ver de qué disco concreto hay que leer, escribir, etc.).

Ahora puedes entender la salida de este comando:

```
unix$ ls -l /dev/tty
crw-rw-rw-  1 root  wheel    2,   0 Aug 20 12:30 /dev/tty
```

El primer carácter es una "`c`", lo que indica *dispositivo de caracteres*. Y los números `2` y `0` corresponden al número principal y secundario.

Los dispositivos sólo son ficheros que representan a los dispositivos, no son hardware. Esto es, que tengas en `/dev` mil dispositivos para discos duros no quiere decir que tengas mil discos instalados.

Para crear un fichero de dispositivo podemos utilizar *mknod(1)*, que llama a *mknod(2)*. Pero, normalmente hay que ser *root* para poderlo hacer. !Así que vamos a convertirnos en *root* y crear uno!:

```
unix$ sudo su
Password:
unix# mknod term c 2 0
sh-3.2# echo hola >term
hola
unix# ls -l term
crw-r--r--  1 root  staff    2,   0 Aug 21 19:56 term
unix# rm term
unix# exit
unix$
```

Como verás, hemos creado un fichero llamado `term`, que es un dispositivo de modo carácter ("`c`") y tiene números `2` y `0` como *major* y *minor*. Si recuerdas el listado de `/dev/tty`, verás que estamos creando un dispositivo para el mismo artefacto. Una vez creado, podemos utilizar `term` igual que `/dev/tty`, como puedes ver por el efecto de `echo` en nuestro ejemplo.