PET: Device Files Explained

Paul A. Marshall

1 Introduction

This PET explains a little about device special files (referred to as device files from now on). What am I talking about? Take a look in your /dev directory.

2 What is a Device File?

You may have heard it said that "eveything in UNIX is a file". Devices are a case in point. Device files are an abstraction provided by the operating system. They hide the real nature of devices from the end user and from applications. With device files, device I/O can be carried out as though we were opening, closing, reading from, or writing to, a regular file.

Take for example, these two instances of the "tar" command:

```
tar cvf /backups/paul3.tar /home/paul/
tar cvf /dev/st0 /home/paul/
```

The first example will package the contents of /home/paul into one neat bundle being the file "/backups/paul3.tar". The second is intended to backup the same contents to tape. The output from tar is the same in both cases. However, in the second case, the kernel will step in and invoke the correct device driver to operate the tape drive. "/dev/st0" is an example of a device file.

3 Let's Take a Look at Them

If I type "`ls -l /dev`" on my system, I get the following output (yes, my system is feeling slightly sick today):

```
lrwxrwxrwx    1 root     root           13 Nov 26  2001 MAKEDEV -> /sbin/MAKEDEV
brw-rw----    1 root     floppy     2,   0 Jan 30  2002 fd0
brw-rw----    1 root     floppy     2,   1 Jul  5  2000 fd1
brw-rw----    1 root     disk       3,   0 Jul  5  2000 hda
brw-rw----    1 root     disk       3,   1 Jul  5  2000 hda1
brw-rw----    1 root     disk       3,   2 Jul  5  2000 hda2
brw-rw----    1 root     disk       3,   3 Jul  5  2000 hda3
brw-rw----    1 root     disk       3,   4 Jul  5  2000 hda4
brw-rw----    1 root     disk       3,  64 Jul  5  2000 hdb
brw-rw----    1 root     disk       3,  65 Jul  5  2000 hdb1
brw-rw----    1 root     disk       3,  66 Jul  5  2000 hdb2
brw-rw----    1 root     disk       3,  67 Jul  5  2000 hdb3
```

```
brw-rw----    1 root     disk       3,  68 Jul  5  2000 hdb4
crw-rw----    1 root     lp         6,   0 Jul  5  2000 lp0
crw-rw----    1 root     lp         6,   1 Jul  5  2000 lp1
crw-rw-rw-    1 root     root       1,   3 Mar 29 23:49 null
crw-------    1 root     dip      108,   0 Dec 27  2001 ppp
crw-rw-rw-    1 root     tty        2, 176 Jul  5  2000 ptya0
crw-rw-rw-    1 root     tty        2, 177 Jul  5  2000 ptya1
crw-rw-rw-    1 root     root       1,   8 Mar 29 23:49 random
crw-rw----    1 root     tape       9,   0 Jul  5  2000 st0
crw-rw----    1 root     tape       9,  96 Jul  5  2000 st0a
crw-rw----    1 root     tape       9,  32 Jul  5  2000 st0l
crw-rw----    1 root     tape       9,  64 Jul  5  2000 st0m
crw-------    1 root     tty        4,   0 Mar 29 23:49 tty0
crw-------    1 root     tty        4,   1 Aug  2 19:39 tty1
crw-rw----    1 root     dialout    4,  64 Jan 15  2002 ttyS0
crw-rw----    1 root     dialout    4,  65 Jul  5  2000 ttyS1
cr--r--r--    1 root     root       1,   9 Aug  2 19:39 urandom
crw-rw-rw-    1 root     root       1,   5 Mar 29 23:49 zero
```

In the first column we have a letter that denotes the file type followed by permissions flags. Mostly, you will see files of type "c" or "b". In the fifth and sixth columns we have some values which are known as *major numbers* and *minor numbers* respectively.

4 Device File Types

Those prefixed with "c" are for character devices. These include things like printers, terminals and serial ports. Those prefixed with "b" are for block devices and include things like hard disk drives, floppy drives and CD drives. Character devices can deal with unbuffered data (ie one byte at a time), whilst block devices must be read from or written to in multiples of their block size (typically 512 or 1024 bytes for a hard disk).

5 Major and Minor Numbers

Ok, so we know that the kernel invokes the appropriate device driver when we access a device file, but how does it know which driver to use? The names of the device files mean nothing to the kernel (funnily enough, they don't mean much to humans either ;-)). Step forward the major numbers. The major number of a device file, together with it's type, is what the kernel will use to look up the driver required to handle things. This may in turn lead to a kernel module being loaded if the driver is not already present in memory (kernel modules would deserve their own PET). If you look at the listing of /dev above, you will notice that devices that require the use of the same driver share the same major number.

The minor number is usually a particular instance of that device type. The kernel simply passes the minor number to the driver and lets it deal with it, so the exact interpretation of the minor number is driver specific.

It is worth noting that major and minor numbers both occupy a single byte. In the case of minor numbers, this means we have a limit of 256 instances of one particular type of device, which would seem to be ample. However, in the case of major numbers, this means we have enough for 256 different types of block device and only 256 different types of character device, quite limiting!

A full list of the major/minor numbers recognised by the kernel can be found if you have the kernel source installed, see ".../Documentation/devices.txt".

6 Not Every Device File Refers to a Real Device

Have you ever spent three hours hunting around your motherboard looking for that pesky null? Me neither. As described in the last section, accessing a device file will lead to the invocation of some kernel routine. There is nothing in the rulebook to say that routine actually has to operate a device. There are a bunch of device files on your system that are not associated with real devices but do other useful things. Here are a few:

/dev/null - perhaps the most well known, if only because UNIX users take great pleasure in using it (and so they should :-)). Anything written to this device will disappear forever. Doesn't sound useful? Sometimes you can't prevent streams of data being produced, but you can control where they go. If you have no interest in seeing it, printing it, storing it for later or anything at all, then /dev/null is a good place to put it. Inside every /dev/null lives a huge purple monster who devours anything and everything on sight. Well, at least, that's a pet theory of mine. You can choose to believe it or not.

/dev/zero - if you read from this file, you will get a stream of zero value bytes. I have only ever used this device to clean hard drives ready for some other use (ie by overwriting with zeroes).

/dev/random - if you read from this, it will spew random bytes at you. This is the kernel's random number generator. Why does the kernel have a random number generator? It has been provided with cryptography in mind. Don't ask me how it works exactly, but the kernel has access to the states of various things and is able to use this information to help produce what are, hopefully, truly random numbers (this is safer than using a purely mathematical algorithm which will produce potentially "crackable" sequences).

7 How to Create Device Files

The *mknod* command is used to create new device files. It is used like this (as superuser):

```
mknod -m <permissions> <name of file> <type b/c> <major no.> <minor no.>
```

For example, to create the fd1 file as shown in the listing above:

```
mknod -m 660 /dev/fd1 b 2 1
```

It is unlikely that you will need to do this very often, but reasons might include:

- Enabling the use of some new whiz bang driver or kernel feature.
- Creating an alias for an existing device file (although it's probably more common practice to create a symlink)

You can create device files wherever you like, but by convention they live in /dev (and if you want your applications to find them, best stick with convention!). Many distributions include a script in the /dev/ directory called "MAKEDEV" which makes light work of creating new device files. I will not go into much detail here as I do not know how standard they are. However, if you find yourself creating a whole raft of new device files, or you simply want to create a missing device file without having to know it's major and minor numbers, take a look at this script.

8 A Few Words About Security

Do not be too cavalier about setting permissions on your device files. If you are not sure about the consequences of widening permissions, then it is probably best not to do it. For example, if an ordinary user is able to raw read from (or even worse, write to) your hard drive device files, it gives them the ability to circumvent your filesystem. If raw read access is allowed, then it allows a slightly skilled user to view the contents of all files therein, regardless of permissions. If write access is allowed, it allows a user the ability to trash your system, skilled or not.

9 Evil Tricks are Possible

Naturally, you do not allow root access to just anyone on *your* box, but maybe you will find yourself using a system where you have no say in the matter. The ability to manipulate device files is probably quite appealing to practical jokers. Just imagine, say, if the major and minor numbers of the device file for your printer were changed to those of /dev/null. It's not particularly clever, but it's subtle enough that it may leave you scratching your head for quite a while. I will leave it at that, I am sure you can think of far worse scenarios.

10 Devfs

This PET would not be complete without mentioning *devfs*. Devfs is an alternative to the system of device files being held on the *real* file system. At the time of writing it is not installed by default by any of the major distributions (AFAIK) and is still considered experimental. Devfs is a virtual file system (like proc) in which device names are listed that have been registered with the kernel. Only devices that are actually present on your system are listed (as opposed to the huge numbers of mostly redundant device files that get created during a normal install). There is no need for major and minor numbers, devfs is a name based system.