Devfs: Concepts, Structures and Practical Applications

Device Parking

Devfs provides a virtual directory tree to organize and tidy up the chaotic "/dev" directory with its numerous static entries. Gentoo Linux and Mandrake already use it, others can now follow suit. We take a look at the benefits that it can bring to both a user and a developer. BY STEFAN KLETT

In the beginning was the File. This is the way UNIX and Linux systems read and write information. Even access to the machine's hardware is handled by special files, or named pipes, which traditionally reside in the "/dev" directory on the root partition.

The sheer bulk of application fields for Linux has seen a tremendous increase in the number of device nodes (created by the "/sbin/MKDEV" script) in the "/dev" directory. It is not uncommon for modern Linux distributions to have about 2000 entries that take up a large proportion of the root filesystem.

Most of these entries are purely speculative. If you take a close look at the "/dev" directory on a Linux machine, you can easily see that only a small proportion of the device nodes are actually used to run system components. How many PC users with Atari mice, 20 hard disks and a RAID system do you know?

This jumble of nodes is more-or-less managed by means of a numerical scheme that assigns major numbers to the hardware driver and minor numbers to the current instance, such as the first of multiple hard disks. Things are starting to look tough for the hard disk based "/dev" mechanism: it seems obvious that the complexity and numbers of supported hardware will continue to increase, instead of becoming more homologous.

Experiments with USB & Co.

Over the past few years a few attempts have been made to overcome the disadvantages of the current static scheme for certain subsystems. These mainly involved subsystems that need to manage dynamically customizable device configurations, such as USB, SCSI, and opening pseudo-terminals for user sessions (devptfs).

Devfs (device filesystem) became a standard component in the 2.3.46 Linux kernel and provides one approach to solving this problem. As the name suggests, a filesystem replaces the functionality of the "/dev" directory. This article shows the approach devfs uses to cope with increasing demand, and what advantages this approach provides. Additionally, we will be introducing functions and drivers that support dynamic device registration in devfs.

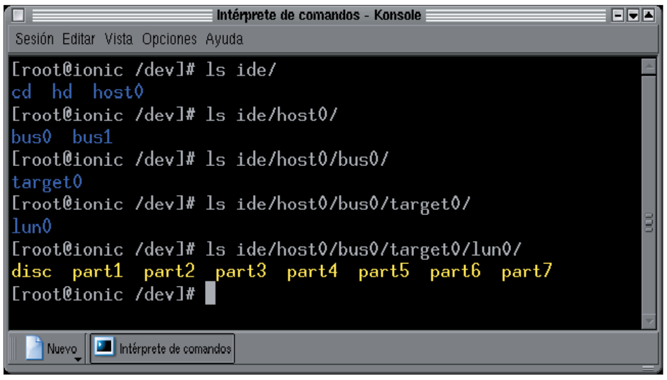
Organized

It is a well-known fact that Linux supports a large variety of filesystems that use a set of primitives provided by VFS (Virtual File System) to describe themselves to the kernel. These include

The Generic Data Model

Filesystems use the following generic data structures to create Linux VFS primitives:

- Superblocks comprise information for managing mounted filesystems, including their type and an indication of whether the filesystem has been written to since mounting.
- Inodes contain information on individual files. The inode number is used internally to identify a file.
- File objects (files) contain information on the interactions between processes and files. File objects only exist within kernel memory while a process is accessing a file.
- Directory entries (dentry) links files and directories by representing access path components. A special buffer called the Dentry Cache is used for this purpose.


Intérprete de comandos - Konsole
Sesión Editar Vista Opciones Ayuda
Nuevo
Intérprete de comandos

```
[root@ionic /dev]# ls ide/
cd  hd  host0
[root@ionic /dev]# ls ide/host0/
bus0  bus1
[root@ionic /dev]# ls ide/host0/bus0/
target0
[root@ionic /dev]# ls ide/host0/bus0/target0/
lun0
[root@ionic /dev]# ls ide/host0/bus0/target0/lun0/
disc  part1  part2  part3  part4  part5  part6  part7
[root@ionic /dev]#
```

Figure 1: The new identification scheme uses full names instead of numbers and single digit names. The tree structure is well-suited to describing hardware specifics

superblocks, inodes, file and directory entries (see the "Generic Data Model" box above).

Devfs is an example of a virtual filesystem, just like procfs and autofs. Virtual filesystems are not persistently stored on a physical medium; instead, the required entries are dynamically created and destroyed in memory at runtime by the kernel. Thus, only those entries actually required exist at any time – in the case of devfs these are the devices currently in use. This reduces the exorbitant number of entries in comparison to the "/dev" mechanism by a considerable amount.

Another effect that devfs has is to reduce the time required to search for devices – one example would be a CD burning program such as KonCD creating a list of the CD ROM drives on the system.

Names and Paths

The directory structure that devfs dynamically manages at run time provides additional benefits. The paths and names used represent the existing hardware and are in a well organised structured – although it may take some getting used to at first. For example, the third partition on the first hard disk, attached to the first IDE controller is referenced by devfs as "/dev/ide/host0/bus0/target0/part3".

Full names provide more information to the operating system, applications and particularly the user, allowing more intuitive and structured navigation in the device arena (see Figure 1). At the same time this removes the need for the arcane major-minor scheme and thus the need to register your own devices with the maintainer of the "devices.txt" file distributed with the kernel. As we will see later, devfs itself is responsible for registering these drivers and clearing them away after they have been used.

Traditional

New names also mean new problems: it is easy to understand that current drivers and programs that access the device files directly will not have any knowledge of the new structure. Thus, Linux systems are currently forced to create devfs compatibility links to ensure access to the hardware using conventional names.

Lock file naming causes similar issues. Designed to ensure exclusive use of devices, filenames in the "/var/lock" directory traditionally adhered to the "LCK.devicename" or "LCK.Major#.Minor#" scheme. As the end nodes in the devfs directory structure often have the same names – typically a number, as in "/dev/usb/tts/0" and "/dev/tts/0" – ambiguity and the drawbacks it can cause cannot be ruled out.

This gives rise to questions on user and access privilege management for the virtual filesystem; simultaneous access by other users is often undesirable for a device being accessed by one user. Most systems will thus decide to implement the devfs daemon, devfsd (see below).

Driver processes can notify the background process which then passes the user ID and access privileges of the calling process to the device in question. Wherever this makes sense, read or write access privileges are assigned exclusively to the original user, thus preventing unwanted competitive access to the device. Alternatively, the required device nodes can be created statically using "mknod" in the init script, and assigning privileges by calls to "chmod" and "chgrp". This approach is useful in minimally interactive environments, where yet another daemon would be too much of a good thing.

One good thing about devfs is the fact that, as a fully-fledged filesystem, it is independent of root filesystem properties. Thus, filesystems that use NTFS, DOS FAT, ISO 9660, or even a ROM as their root partition are viable, despite being incapable of managing symbolic links and named pipes.

Disk based device access prevented this approach previously. If the UID of a ptty pseudo-terminal changes from "root" to the real user, the inode for the ptty device node is written to – of course, this would be impossible on a read-only filesystem. As a more generic solution, devfs completely assumes the role of devptsfs, as required by the Unix98 standard.

Daemonic Manager

As devfs dynamically customizes the device entries to reflect the current environment, there must be a facility for this purpose within devfs.

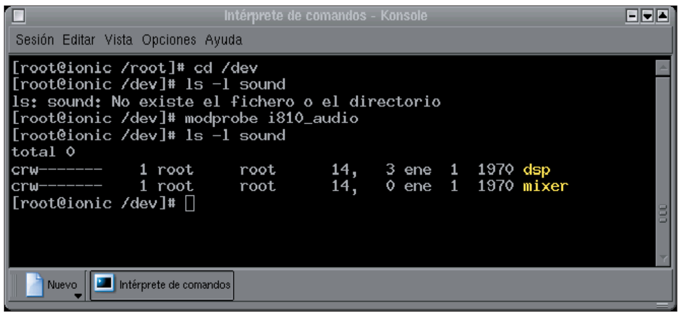
The devfsd process is normally best suited to this task. Devfsd is called by the "rc.sysinit" script when a machine is bootstrapped. In addition to automatically customizing UIDs and setting user privileges, devfsd automatically creates any symbolic links required for downward compatibility.

The Linux administrator can also use the "/etc/devfsd.conf" file, to explicitly tell devfsd to create user-defined links. The practical thing about this is that regular expressions can be used to define ‘allow’ or ‘deny’ rules for user-definable actions in “devfsd.conf”; the “devfsd.conf” manpage provides more details on the syntax.

Normally, a hardware driver registers by calling its internal “devfs_register” function to let devfsd know that it has been loaded. The “try_modload” devfs function, which informs devfsd whenever an application attempts to access the inode of a device entry, is responsible for on-the-fly module loading. Devfsd then creates a entry that reflects the UID and privileges of the calling process, unless it has been configured not to do so (see Figure 2).

Finally, devfsd ensures that any changes to access privileges persist after rebooting. To do so, it copies the inodes for any changed entries to the directory defined in “/etc/devfsd.conf”. When the system reboots, devfsd restores the inodes by writing them back.

Devfs does not need to use “/dev” as a mountpoint and can even be mounted multiple times. Thus, you need to define the root directory for devfsd on calling the daemon. Multiple devfsd instances can thus manage multiple device file system images in parallel.


Intérprete de comandos - Konsole
Sesión Editar Vista Opciones Ayuda
Nuevo
Intérprete de comandos

```
[root@ionic /root]# cd /dev
[root@ionic /dev]# ls -l sound
ls: sound: No existe el fichero o el directorio
[root@ionic /dev]# modprobe i810_audio
[root@ionic /dev]# ls -l sound
total 0
crw-------    1 root     root      14,   3 ene  1  1970 dsp
crw-------    1 root     root      14,   0 ene  1  1970 mixer
[root@ionic /dev]#
```

Figure 2: Demonstration of the registration function. When a module is loaded, devfs automatically creates a directory entry and leaf nodes

Customizing Drivers for Devfs

To allow a driver to work with devfs, you must import the “devfs_fs_kernel.h” kernel file. Amongst other things, the file defines the “devfs_register” and “devfs_unregister” functions, as shown in Listing 1. As the names suggest, these modules are responsible for controlling the registration of devfs entries, and use “devfs_handle” as a pointer to a “devfs_entry” structure.

The parameters passed to “devfs_register” exactly match the definable components of the record type. A number of list pointers are provided to handle management tasks, however, driver developers do not need to concern themselves with them.

The first parameter specifies the directory where the new device will be stored, such as “/dev/ide/”. Currently, an additional function, “devfs_mkdir”, is responsible for registering directories – if you intend to support a new device class, that is. “devfs_mkdir” also returns a “devfs _handle”:

```
devfs_handle_t devfs_mk_dir ➥
(devfs_handle_t dir, const ➥
char *name, void *info)
```

The “flags” parameter specifies whether the entry automatically inherits the UID of the calling process or the identity of the user that created the entry manually, such as “root” for example. The range of values for the “flags” parameter is listed in “devfs_fs _kernel.h”. “flags” can even hide an entry, preventing it from being displayed when the directory entries are listed. In fact, this is how removable devices are handled during runtime.

Devfs may make major/minor numbers unnecessary, but it is still important to make sure that applications based on the old scheme will be able to coexist with the new. To this end, devfs entries contain variables to identify themselves to the numbering scheme. Major and minor numbers are provided by the “devfs_register_chardev” and “devfs_register_blkdev” wrappers for example; the wrappers prevent the “devfs-icized” versions of the same name from being called. This allows you to call “mknod” to create appropriate entries.

There are also symbolic link functions that automatically create compatibility links when a module is loaded. The “mode_t” type is an integer that specifies the UID and access privileges to be applied when creating an entry.

“ops” refers to a “file_operations” structure from “linux/include/fs/fs.h” that allows you to define valid access operations for an entry. “Info” has no meaning for most modules, and is typically set to “NULL”. The functions introduced at this point allow you to enhance and add functions. The sourcefile “linux/fs/devfs/base.c” provides further details.

Practical Devfs Applications

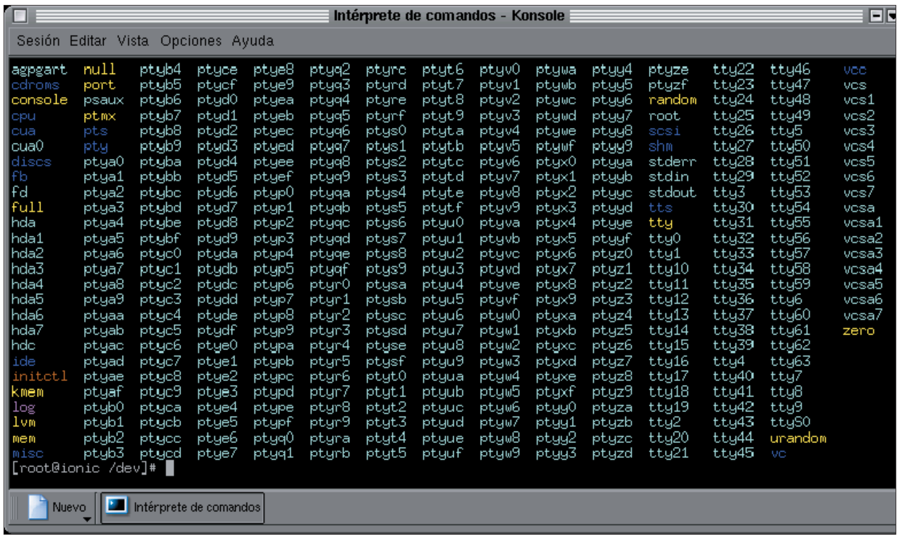
To perform non-invasive testing of a devfs based system, you might like to download the ISO image of the Gentoo Linux Live CD from [4]. Incidentally, this system leverages the fact that the CD ROM can be used as a universal root filesystem. Thus, you do not need a free partition on your hard disk.

Expect your mileage to differ, if you attempt to enable devfs on other Linux distributions. Our test candidate for this article was Conectiva 8.0, although we had to replace the default 2.4.18 kernel with a 2.4.20 version, as we were unable to compile the Conectiva sources with devfs functionality.

The 2.4.20 kernel, that is also used on the Gentoo CD, did not cause us any trouble after enabling the devfs configuration options:

Listing 1: Excerpt from “devfs_fs_kernel.h”

```
extern devfs_handle_t devfs_register     (devfs_handle_t dir, const char *name, unsigned int flags,
                                          unsigned int major, unsigned int minor, umode_t mode,
                                          void *ops, void *info);
extern void devfs_unregister (devfs_handle_t de);
```


Intérprete de comandos - Konsole
Sesión Editar Vista Opciones Ayuda
Nuevo
Intérprete de comandos

```
agpgart  null     ptyb4  ptyce  ptye8  ptyq2  ptyrc  ptyt6  ptyv0  ptywa  ptyy4  ptyze    tty22  tty46    vcc
cdroms   port     ptyb5  ptycf  ptye9  ptyq3  ptyrd  ptyt7  ptyv1  ptywb  ptyy5  ptyzf    tty23  tty47    vcs
console  psaux    ptyb6  ptyd0  ptyea  ptyq4  ptyre  ptyt8  ptyv2  ptywc  ptyy6  random   tty24  tty48    vcs1
cpu      ptmx     ptyb7  ptyd1  ptyeb  ptyq5  ptyrf  ptyt9  ptyv3  ptywd  ptyy7  root     tty25  tty49    vcs2
cua      pts      ptyb8  ptyd2  ptyec  ptyq6  ptys0  ptyta  ptyv4  ptywe  ptyy8  scsi     tty26  tty5     vcs3
cua0     pty      ptyb9  ptyd3  ptyed  ptyq7  ptys1  ptytb  ptyv5  ptywf  ptyy9  shm      tty27  tty50    vcs4
discs    ptya0    ptyba  ptyd4  ptyee  ptyq8  ptys2  ptytc  ptyv6  ptyx0  ptyya  stderr   tty28  tty51    vcs5
fb       ptya1    ptybb  ptyd5  ptyef  ptyq9  ptys3  ptytd  ptyv7  ptyx1  ptyyb  stdin    tty29  tty52    vcs6
fd       ptya2    ptybc  ptyd6  ptyp0  ptyqa  ptys4  ptyte  ptyv8  ptyx2  ptyyc  stdout   tty3   tty53    vcs7
full     ptya3    ptybd  ptyd7  ptyp1  ptyqb  ptys5  ptytf  ptyv9  ptyx3  ptyyd  tts      tty30  tty54    vcsa
hda      ptya4    ptybe  ptyd8  ptyp2  ptyqc  ptys6  ptyu0  ptyva  ptyx4  ptyye  tty      tty31  tty55    vcsa1
hda1     ptya5    ptybf  ptyd9  ptyp3  ptyqd  ptys7  ptyu1  ptyvb  ptyx5  ptyyf  tty0     tty32  tty56    vcsa2
hda2     ptya6    ptyc0  ptyda  ptyp4  ptyqe  ptys8  ptyu2  ptyvc  ptyx6  ptyz0  tty1     tty33  tty57    vcsa3
hda3     ptya7    ptyc1  ptydb  ptyp5  ptyqf  ptys9  ptyu3  ptyvd  ptyx7  ptyz1  tty10    tty34  tty58    vcsa4
hda4     ptya8    ptyc2  ptydc  ptyp6  ptyr0  ptysa  ptyu4  ptyve  ptyx8  ptyz2  tty11    tty35  tty59    vcsa5
hda5     ptya9    ptyc3  ptydd  ptyp7  ptyr1  ptysb  ptyu5  ptyvf  ptyx9  ptyz3  tty12    tty36  tty6     vcsa6
hda6     ptyaa    ptyc4  ptyde  ptyp8  ptyr2  ptysc  ptyu6  ptyw0  ptyxa  ptyz4  tty13    tty37  tty60    vcsa7
hda7     ptyab    ptyc5  ptydf  ptyp9  ptyr3  ptysd  ptyu7  ptyw1  ptyxb  ptyz5  tty14    tty38  tty61    zero
hdc      ptyac    ptyc6  ptye0  ptypa  ptyr4  ptyse  ptyu8  ptyw2  ptyxc  ptyz6  tty15    tty39  tty62
ide      ptyad    ptyc7  ptye1  ptypb  ptyr5  ptysf  ptyu9  ptyw3  ptyxd  ptyz7  tty16    tty4   tty63
initctl  ptyae    ptyc8  ptye2  ptypc  ptyr6  ptyt0  ptyua  ptyw4  ptyxe  ptyz8  tty17    tty40  tty7
kmem     ptyaf    ptyc9  ptye3  ptypd  ptyr7  ptyt1  ptyub  ptyw5  ptyxf  ptyz9  tty18    tty41  tty8
log      ptyb0    ptyca  ptye4  ptype  ptyr8  ptyt2  ptyuc  ptyw6  ptyy0  ptyza  tty19    tty42  tty9
lvm      ptyb1    ptycb  ptye5  ptypf  ptyr9  ptyt3  ptyud  ptyw7  ptyy1  ptyzb  tty2     tty43  ttyS0
mem      ptyb2    ptycc  ptye6  ptyq0  ptyra  ptyt4  ptyue  ptyw8  ptyy2  ptyzc  tty20    tty44  urandom
misc     ptyb3    ptycd  ptye7  ptyq1  ptyrb  ptyt5  ptyuf  ptyw9  ptyy3  ptyzd  tty21    tty45  vc
[root@ionic /dev]#
```

Figure 3: The new structure and the relatively small number of entries at the top level. Many of these are simply links designed to ensure backward compatibility

- "CONFIG_EXPERIMENTAL",
- "CONFIG _DEVFS_FS", and
- "CONFIG_DEVFS _MOUNT".

Following this, we downloaded the sources from devfs developer, Richard Gooch's homepage [5], and then went on to compile and install. "make, make install" was more or less all we needed.

As the file [1] distributed with the kernel sources explains in detail, there were a few more steps to complete to set up the configuration files on the test system.

First, we had to copy the default "devfsd.conf", supplied with the sources, to "/etc". Then we set up "/etc/rc.d/sysinit.d" to launch devfsd by adding a "/sbin/devfsd/dev" line. This enables devfs as early as possible in the boot process. This was not necessary for Conectiva Linux, as the entry already existed.

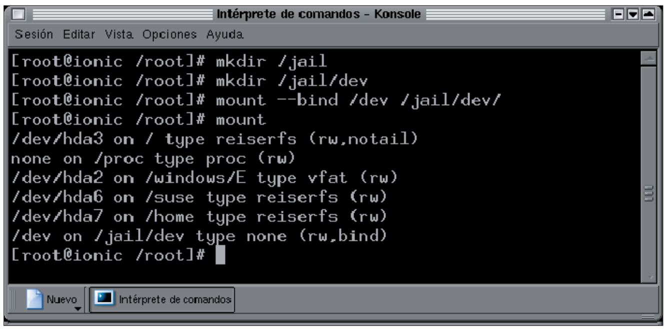
To allow PAM (Pluggable Authentication Modules) to work with devfs, you have to enter the new names for the local virtual consoles in "/etc/securetty" to allow root logins on the consoles:

```
vc/1
vc/2
...
vc/8
```

This step is required, as PAM does not accept symbolic links for the console. You can comment out the "/dev/pts" entry in "/etc/fstab", as it is no longer needed with devfs. This completes the preparatory work – it is now time to boot the machine with the new kernel.

In our lab the boot process went quite smoothly with the exception of one or two error messages caused by drivers that needed customizing. After logging on to the machine, the new, and eagerly awaited, structure of "/dev" was revealed (see Figure 3).

Figure 4 shows the humble beginnings of setting up a chroot jail with devfs. This is fairly easy to handle as devfs inherits the capability of using multiple mountpoints simultaneously from Linux VFS. To restrict access to individual nodes, you can mount individual subdirectories of the "/dev" hierarchy, by using "devfsd.conf" entries to remove device nodes from the registry.


Intérprete de comandos - Konsole
Sesión Editar Vista Opciones Ayuda
Nuevo
Intérprete de comandos

```
[root@ionic /root]# mkdir /jail
[root@ionic /root]# mkdir /jail/dev
[root@ionic /root]# mount --bind /dev /jail/dev/
[root@ionic /root]# mount
/dev/hda3 on / type reiserfs (rw,notail)
none on /proc type proc (rw)
/dev/hda2 on /windows/E type vfat (rw)
/dev/hda6 on /suse type reiserfs (rw)
/dev/hda7 on /home type reiserfs (rw)
/dev on /jail/dev type none (rw,bind)
[root@ionic /root]#
```

Figure 4: Initial steps for creating a chroot jail with devfs

Conclusion

Devfs is a well-balanced concept, and equally well-suited to eliminating the chaos on the root disk.

As it has the properties of a filesystem, devfs handles many aspects more cleanly and elegantly than the older "/dev" directory scheme with its innumerable static entries. Despite this, devfs obviously retains the UNIX paradigm that "everything is a file".

Although driver support is still missing in some cases, drivers should be fairly easy to program, as this article has hopefully demonstrated. Symbolic compatibility links can be used to trick applications and tools that cannot handle the new device names into thinking that the old "/dev" system still exists.

The fact that distributions such as Gentoo Linux and Mandrake 9.0 have already moved to devfs, demonstrates its practical use under Linux.

If other distributors like SuSE and Red Hat decide to get on board the devfs train, Linux developers around the world would be forced to provide the new devfs support for their programs. This would remove the need for compatibility code and additional links and so in turn, allow devfs to demonstrate its true value to the user. ■

THE AUTHOR

Stefan Klett still studies computer science at Karlsruhe University. Currently he is working on his student thesis on active networks based on iptables. Apart from his studies he is working in network administration and programming. He has already contributed to several Linux publications.

INFO

[1] Devfs basics: "Documentation/filesystems/devfs/README" in the kernel sources

[2] The source of all sources: *http://lxr.linux.no/source/fs/devfs/*

[3] Manpage for devfsd: *http://www.fifi.org/cgi-bin/man2html/usr/share/man/man8/devfsd.8.gz*

[4] ISO image for Gentoo Linux: *http://www.gentoo.org/main/en/where.xml*

[5] Devfs sources: *http://www.atnf.csiro.au/~rgooch/linux/*