

## Kernel Korner

# udev--Persistent Device Naming in User Space



*Whether you're plugging a camera and scanner in to your laptop or adding another SCSI drive to your company server, it's time to end the current mess of major and minor numbers.*

*by Greg Kroah-Hartman*

Starting with the 2.5 kernel, all physical and virtual devices in a system are visible to user space in a hierarchal fashion through sysfs. /sbin/hotplug provides a notification to user space when any device is added or removed from the system. Using these two features, a user-space implementation of a dynamic /dev now is possible that can provide a flexible device naming policy.

This article discusses udev, a program that replaces the functionality of devfs. It provides /dev entries for devices in the system at any moment in time. It also provides features previously unavailable through devfs alone, such as persistent naming for devices when they move around the device tree, a flexible device naming scheme, notification of external systems of device changes and moving all naming policy out of the kernel.

The /dev directory on a Linux machine is where all of the device files for the system should be located. A device file details how a user program can access a specific hardware device or function. For example, the device file /dev/hda traditionally is used to represent the first IDE drive in the system. The name hda corresponds to both a major and a minor number, which are used by the kernel to determine what hardware device to talk to. Currently, a wide range of names that match up to different major and minor numbers has been defined.

All major and minor numbers are assigned a name that matches up with a type of device. This allocation is done by the Linux assigned names and numbers authority (LANANA), and the current device list can be found on its Web site (see the on-line Resources section).

As Linux begins supporting new kinds of devices, these devices need to be assigned a major and minor number range in order for users to access them through the /dev directory. An alternative is to provide access through a filesystem; my 2002 linux.conf.au paper provides more details on how to do this (see the on-line Resources section). In kernel versions 2.4 and earlier, the valid range of major numbers was 1–255, and the valid range of minor numbers was 1–255. Because of this limited range, a freeze was placed on allocating new major and minor numbers during the 2.3 development cycle. This freeze since has been lifted, and the 2.6 kernel had the valid range of major numbers increased to 4,095. More than a million minor numbers are available per major number.

## What /dev Entry Is Which Device

When the kernel finds a new piece of hardware, it typically assigns the next major/minor pair for that kind of hardware to the device. So, on boot, the first USB printer found would be assigned the major number 180 and minor number 0, which is referenced in /dev as /dev/usb/lp0. The second USB printer would be assigned major number 180 and minor number 1, which is referenced in /dev as /dev/usb/lp1. If the user rearranges the USB topology, perhaps adding a USB hub to support more USB devices in the system, the USB probing

order of the printers might change the next time the computer is booted, reversing the assignment of the different minor numbers to the two printers.

This same situation holds true for almost any kind of device that can be removed or added while the computer is powered up. With the advent of PCI hot-plug-enabled systems and hot-pluggable buses, such as IEEE 1394, USB and CardBus, almost all devices have this problem.

With the advent of the sysfs filesystem in the 2.5 kernel, the problem of determining which device minor number is assigned to which physical device is much easier to determine. For a system having two different USB printers plugged in to it, the sysfs /sys/class/usb directory tree would look like this:

```
/sys/class/usb/
|-- lp0
|   |-- dev
|   |-- device -> ../../../../devices/pci0/00:09.0/usb1/1-1/1-1:0
|   `-- driver -> ../../../../bus/usb/drivers/usblp
`-- lp1
    |-- dev
    |-- device -> ../../../../devices/pci0/00:0d.0/usb3/3-1/3-1:0
    `-- driver -> ../../../../bus/usb/drivers/usblp

$ cat /sys/class/usb/lp0/device/serial
HXOLL0012202323480
$ cat /sys/class/usb/lp1/device/serial
W09090207101241330
```

Within the individual USB device directories pointed to by the lp0/device and lp1/device symbolic links, a lot of USB-specific information can be determined, such as the manufacturer of the device and the (hopefully unique) serial number.

As can be seen by the serial files in the above description, the /dev/usb/lp0 device file is associated with the USB printer with serial number HXOLL0012202323480, and the /dev/usb/lp1 device file is associated with the USB printer with serial number W09090207101241330. If these printers are moved around, say by placing them both behind a USB hub, they might be renamed, as they are probed in a different order on startup:

```
$ tree /sys/class/usb/
/sys/class/usb/
|-- lp0
|   |-- dev
|   |-- device -> ../../../../devices/pci0/00:09.0/usb1/1-1/1-1.1/1-1.1:0
|   `-- driver -> ../../../../bus/usb/drivers/usblp
`-- lp1
    |-- dev
    |-- device -> ../../../../devices/pci0/00:09.0/usb1/1-1/1-1.4/1-1.4:0
    `-- driver -> ../../../../bus/usb/drivers/usblp

$ cat /sys/class/usb/lp0/device/serial
W09090207101241330
$ cat /sys/class/usb/lp1/device/serial
HXOLL0012202323480
```

As this description shows, the /dev/usb/lp0 device now is assigned to the USB printer with the serial number W09090207101241330 due to this different probing order.

sysfs enables a user to determine which device has been assigned by the kernel to which device file. This is a

powerful association that previously had not been easily available. However, a user generally does not care that `/dev/usb/lp0` and `/dev/usb/lp1` are now reversed and should be changed in a configuration file somewhere. The user simply wants to be able to print to the proper printer, no matter where it is in the USB device tree.

## **/dev Is Too Big**

Not all device files in the `/dev` directory of most distributions match up to a physical device that is currently connected to the computer. Instead, the `/dev` directory is created when the operating system is initialized on the machine, populating the `/dev` directory with all known possible names. On a machine running Red Hat's Fedora release 1, the `/dev` directory holds more than 18,000 different entries. This many entries soon become unwieldy for users trying to determine exactly what devices currently are present.

## **devfs**

Because of the large numbers of device files in the `/dev` directory, a number of operating systems have moved to having the kernel itself manage the `/dev` directory, as the kernel always knows exactly what devices are present on the system. It does this by creating a RAM-based filesystem called devfs. Linux also has this option, and it has become popular over time in a number of different distributions, including Gentoo.

For a number of people, devfs solves their immediate needs. However, the Linux-based devfs implementation still has a number of unsolved problems. Most notably, it does not provide the ability to create device nodes with a persistent name.

## **udev's goals**

In light of the previously mentioned problems, the udev Project was started. Its goals are to run in user space; create a dynamic `/dev`; provide consistent device naming, if wanted; and provide a user-space API to access information about current system devices. For more on how udev compares with devfs, see the on-line Resources section.

The first item, run in user space, is accomplished by harnessing the fact that `/sbin/hotplug` generates an event for every device added to or removed from the system with sysfs' ability to show all the needed information about all devices.

The second item, create a dynamic `/dev`, is handled by catching all `/sbin/hotplug` events, looking up the major and minor number in sysfs for the added device and creating a `/dev` file with the kernel name the device was assigned. If the device was removed from the system, it is easy to remove the `/dev` entry for that device.

udev achieved these first two goals back in April 2003 in an extremely tiny 6Kb of compiled code, proving that this scheme of catching hot-plug events and using sysfs was feasible and quite simple to implement. Since that humble beginning in early 2003, udev has achieved all of its goals. It provides users with the ability to name devices in a persistent manner using a flexible rule-based system.

udev's rules are contained in the `/etc/udev/udev.rules` file and describe any devices the user wants to name in a way that differs from the default kernel name. Here's an example of a udev.rules file:

```
# if /sbin/scsi_id returns "OEM 0815" device will
# be called disk1
BUS="scsi", PROGRAM="/sbin/scsi_id", \
```

```

RESULT="OEM 0815", NAME="disk1"

# USB printer to be called lp_color
BUS="usb", SYSFS_serial="W09090207101241330", \
NAME="lp_color"

# SCSI disk with a specific vendor and model number
# is to be called boot
BUS="scsi", SYSFS_vendor="IBM", \
SYSFS_model="ST336", NAME="boot"

# sound card with PCI bus id 00:0b.0 to be called dsp
BUS="pci", ID="00:0b.0", NAME="dsp"

# USB mouse at third port of the second hub to
# be called mouse1
BUS="usb", PLACE="2.3", NAME="mouse1"

# ttyUSB1 should always be called pda with two
# additional symlinks
KERNEL="ttyUSB1", NAME="pda", \
SYMLINK="palmtop handheld"

# multiple USB webcams with symlinks to be called
# webcam0, webcam1, ...
BUS="usb", SYSFS_model="XV3", NAME="video%n", \
SYMLINK="webcam%n"

```

A udev rule defines the mapping between a device's attributes and the desired device filename. To do this, a number of keys can be queried from the device to determine a match. If no match is found in the udev.rules file, the default kernel name is used. Below is a list of the different types of keys that udev understands:

- **BUS:** matches the bus type of the device; examples of this include PCI, USB or SCSI.
- **KERNEL:** matches the name the kernel gives the device.
- **ID:** matches the device number on the bus; for example, the PCI bus ID or the USB device ID.
- **PLACE:** matches the topological position on bus, such as the physical port a USB device is plugged in to.
- **SYSFS\_filename, SYSFS{filename}:** allows udev to match any sysfs device attribute, such as label, vendor, USB serial number or SCSI UUID. Up to five different sysfs files can be checked in a single rule, with all of the values being required in order to match the rule.
- **PROGRAM:** allows udev to call an external program and check the result. This key is valid if the program returns successfully. The string returned by the program additionally may be matched with the **RESULT** key.
- **RESULT:** matches the returned string of the last **PROGRAM** call. This key may be used in any rule following a **PROGRAM** call.

After the different keys, a **NAME** and optional **SYMLINK** are specified. The **NAME** is what udev uses to call the device if the rule matches, and the **SYMLINK** specifies what, if any, symlinks also are generated. More than one symlink can be specified at once, with spaces between multiple symlinks. Both the **NAME** and **SYMLINK** files can contain directories to allow **/dev** to be simplified.

## Example

So, back to our two-printer example. To name both of these devices in a consistent manner, the following two udev rules might be used:

```
BUS="usb", SYSFS_serial="W09090207101241330", \
NAME="lp_color"
BUS="usb", SYSFS_serial="HXOLL0012202323480", \
NAME="lp_plain"
```

These rules cause udev to look at the sysfs file serial for both printers and, depending on the value in the file, name the printer either `lp_color` or `lp_plain`. This ensures that no matter which device was plugged in first or detected first or whether another USB printer is added to the system, both of the printers have the same persistent name.

## Advanced udev Rules

udev allows a number of printf-like string substitutions to be used in the NAME, SYMLINK and PROGRAM fields in the udev.rules file. These fields are:

- %n: kernel number of the device; for example, the device `sda3` has a kernel number of 3.
- %k: kernel name for the device.
- %M: kernel major number for the device.
- %m: kernel minor number for the device.
- %b: bus ID for the device.
- %c: PROGRAM returned string. A number can be added to this modifier to pick up only a specific word within the string. This field does not work within the PROGRAM field for the obvious reason.
- %%: % character itself.

Also, a number of the different keys support a simple form of shell-style pattern matching. These patterns are:

- \*: matches zero, one or more characters.
- ?: matches any single character but does not match zero characters.
- [ ]: matches any single character specified within the brackets; for example, the pattern string `tty[SR]` would match either `ttyS` or `ttyR`. Ranges also are supported within this match with the `-` character. For example, to match on the range of all digits, the pattern `[0-9]` would be used. If the first character following the `[` is `!`, any character not enclosed is matched.

Because of the ability to do these simple string substitutions and string pattern matching, combined with the ability to have udev run any other program and use its result, udev has become an extremely flexible tool for naming devices. As an example of this power, take a look at the following rule:

```
KERNEL="[hs]d[a-z]", PROGRAM="name_cdrom.pl %M %m", \
NAME="%1c", SYMLINK="cdrom"
```

This rule matches any block device and calls the Perl script `name_cdrom.pl` with the major and minor number of the device. If this program is successful, `udev` uses the first word of the program's output to name the device and creates a symlink called `cdrom`. The `name_cdrom.pl` script can be found in the `udev` release.

What this script does is determine whether the device is a CD-ROM device. If it is, it queries the Free CDDDB database to see if the CD-ROM present in the device is known in the database. If it is, it then names the device based on the CD. For example, my `/dev` looks like the following when using this rule:

```
$ ls -l /dev/S* /dev/cdrom
brw----- 1 root root 22, 64 Feb 15 08:26 /dev/Samiam-Astray
lrwxrwxrwx 1 root root    8 Feb 15 08:26 /dev/cdrom ->
/dev/Samiam-Astray
```

This shows how `udev` can query a database across the Internet to determine how to name a device. Yes, it's a crazy naming scheme to try to use, but it shows how powerful and flexible `udev` can be.

## Acknowledgements

The author would like to thank Daniel Stekloff of IBM, who has helped shape the design of `udev` in many ways. Without his perseverance, `udev` would not be available at all. Also, Kay Sievers has been instrumental in implementing the majority of the advanced features in `udev`, most notably the ability to have string modifiers and pattern matching, both of which are essential for using `udev` in the real world. Without his help, `udev` would not be anywhere near as powerful and useful as it is.

Also, without Pat Mochel's `sysfs` and driver model core, `udev` would not have been possible to implement. The author is indebted to him for undertaking what most thought of as an impossible task and for allowing others to build easily on his common framework, allowing all users to see the “Web woven by a spider on drugs” that the kernel keeps track of.

This article is based on the Ottawa Linux Symposium 2002 paper on `udev` (see Resources).

**Resources for this article:** [www.linuxjournal.com/article/7496](http://www.linuxjournal.com/article/7496).

Greg Kroah-Hartman currently is the Linux kernel maintainer for a variety of different driver subsystems. He works for IBM, doing Linux kernel-related things, and can be reached at [greg@kroah.com](mailto:greg@kroah.com).

---