

About

udev allows Linux users to have a dynamic /dev directory and it provides the ability to have persistent device names.



Documentation

udev comes with a lot of documentation and full manpages included in the source tarball. If, for some reason, that is not enough, here is a list of some online references about the program, and how to configure it properly:

- A [FAQ](#) which answers a number of questions about what udev is, and how it compares to other programs.
- A [OLS 2003 paper](#) about how udev was originally designed, and why it was created.
- The [OLS 2003 presentation](#) that went along with the paper.
- A [more recent article about udev](#) and how it works was published in Linux Journal.
- [Writing udev rules](#) This is the online copy of the same file contained within the udev release.
- [A general udev primer](#)
- [udev on Fedora](#)
- [Gentoo udev guide](#)
- [The last word on how udev and devfs compare.](#)

Requirements

It requires a 2.6 Linux kernel.

Getting

The latest version of udev is available on kernel.org.

Using

udev is included in almost every 2.6 kernel based Linux distribution that is shipping, so please use the packages provided by your distro instead of trying to install from the source tree. But if you insist, please read the [README](#) files in the source tarball for how to set it up initially.

Questions

Any questions about udev should be addressed to the email address located in the [README](#) file within the source tarball.

Thanks

udev was developed by Greg Kroah-Hartman and Kay Sievers. With much help from Dan Stekloff and many others

Frequently Asked Questions about udev

Q: What's this udev thing, and what is it trying to do?

A: Read the OLS 2003 paper about udev, available in the docs/ directory, and at:

http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf

There is also a udev presentation given at OLS 2003 available at:

http://www.kroah.com/linux/talks/ols_2003_udev_talk/

Q: How is udev related to devfs?

A: udev works entirely in userspace, using hotplug events the kernel sends whenever a device is added or removed from the kernel. Details about the devices are exported by the kernel to the sysfs filesystem at /sys. All device naming policy permission control and event handling is done in userspace. devfs is operated from within the kernel.

Q: Why was devfs marked OBSOLETE/removed if udev can't do everything devfs did?

A: To quote Al Viro (Linux VFS kernel maintainer):

- it was determined that the same thing could be done in userspace
- devfs had been shoved into the tree in hope that its quality will catch up
- devfs was found to have fixable and unfixable bugs
- the former had stayed around for many months with maintainer claiming that everything works fine
- the latter had stayed, period.
- the devfs maintainer/author disappeared and stopped maintaining the code.

Q: But udev will not automatically load a driver if a /dev node is opened when it is not present like devfs will do.

A: Right, but Linux is supposed to load a module when a device is discovered not to load a module when it's accessed.

Q: Oh come on, pretty please. It can't be that hard to do.

A: Such a functionality isn't needed on a properly configured system. All devices present on the system should generate hotplug events, loading the appropriate driver, and udev will notice and create the appropriate device node. If you don't want to keep all drivers for your hardware in memory, then use something else to manage your modules (scripts, modules.conf, etc.) This is not a task for udev.

Q: But I love that feature of devfs, please?

A: The devfs approach caused a lot of spurious modprobe attempts as programs probed to see if devices were present or not. Every probe attempt created a process to run modprobe, almost all of which were spurious.

Q: I really like the devfs naming scheme, will udev do that?

A: Yes, udev can create /dev nodes using the devfs naming policy. A configuration file needs to be created to map the kernel default names to the devfs names. See the udev.rules.devfs file in the udev release.

Note that the devfs scheme is not recommended or officially supported cause it is a really stupid idea to simply enumerate devices in a world where devices can come and go at any time. These numbers give you nothing but problems, and are not useful to identify a device. Have a look at the persistent disk rules for an example how to do it correctly in userspace without any stupid device enumeration.

Q: What kinds of devices does udev create nodes for?

A: All devices that are shown in sysfs will work with udev. If more support is added for devices to the kernel, udev will automatically start working for them. All block devices are currently supported, and almost all major char devices are supported. Kernel developers are working on adding support for all char devices at this time. See the linux-kernel mailing list for patches and status of these patches.

Q: Will udev remove the limit on the number of anonymous devices?

A: udev is entirely in userspace. If the kernel supports a greater number of anonymous devices, udev will support it.

Q: Will udev support symlinks?

A: Yes, It now does. Multiple symlinks per device node are supported.

Q: How will udev handle the /dev filesystem?

A: /dev is recommended to be a tmpfs filesystem that is recreated on every reboot. Although, udev does not care what kind of filesystem it runs on.

Q: How will udev handle devices found before init runs?

A: udev can be placed in initramfs and run for every device that is found. udev can also populate an initial /dev directory from the content of /sys after the real root is mounted.

Q: Can I use udev to automount a USB device when I connect it?

A: Technically, yes, but udev is not intended for this. All major distributions use HAL (http://freedesktop.org/wiki/Software_2fhal) for this, which also watches devices with removable media and integrates into the desktop software.

Alternatively, it is easy to add the following to fstab:

```
/dev/disk/by-label/PENDRIVE /media/PENDRIVE vfat user,noauto 0 0
```

This means that users can access the device with:

```
$mount /media/PENDRIVE
```

and doesn't have to be root, but will get full permissions on the device.

Using the persistent disk links (label, uuid) will always catch the same device regardless of the actual kernel name.

Q: Are there any security issues that I should be aware of?

A: When using dynamic device numbers, a given pair of major/minor numbers may point to different hardware over time. If a user has permission to access a specific device node directly and is able to create hard links to this node, he or she can do so to create a copy of the device node. When the device is unplugged and udev removes the device node, the user's copy remains. If the device node is later recreated with different permissions the hard link can still be used to access the device using the old permissions. (The same problem exists when using PAM to change permissions on login.)

The simplest solution is to prevent the creation of hard links by putting /dev in a separate filesystem like tmpfs.

Q: I have other questions about udev, where do I ask them?

A: The linux-hotplug-devel mailing list is the proper place for it. The address for it is linux-hotplug-devel@lists.sourceforge.net

Information on joining can be found at

<<https://lists.sourceforge.net/lists/listinfo/linux-hotplug-devel>>

Archives of the **mailing list can be found at:**

Reprinted from the
**Proceedings of the
Linux Symposium**

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.

udev – A Userspace Implementation of devfs

*Greg Kroah-Hartman**

IBM Corp.

Linux Technology Center

greg@kroah.com, gregkh@us.ibm.com

Abstract

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

This paper will discuss `udev`, a program that replaces the functionality of `devfs` (only providing `/dev` entries for devices that are in the system at any moment in time), and allows for features that were previously not able to be done through `devfs` alone, such as:

- Persistent naming for devices when they move around the device tree.
- Notification of external systems of device changes.
- A flexible device naming scheme.
- Allow the kernel to use dynamic major and minor numbers
- Move all naming policy out of the kernel.

*This work represents the view of the author and does not necessarily represent the view of IBM.

This paper will describe why such a userspace program is superior to a kernel based `devfs`, and detail the design decisions that went into its creation. The paper will also describe how `udev` works, how to write plugins that extend the functionality of it (different naming schemes, etc.), and different trade offs that were made in order to provide a working system.

1 Introduction

The `/dev` directory on a Linux machine is where all of the device files for the system should be located.[2] A device file is how a user program can access a specific hardware device or function. For example, the device file `/dev/hda` is traditionally used to represent the first IDE drive in the system. The name `hda` corresponds to both a major and a minor number, which is used by the kernel to determine what hardware device to talk to. Currently a very wide range of names that match up to different major and minor numbers have 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)[4] and the current device list can be always be found on their web site at <http://www.lanana.org/docs/device-list/devices.txt>

As Linux gains support for new kinds of devices, they need to be assigned a major and minor number range in order for the user to be able to access them through the `/dev` directory (one alternative to this is to provide access through a filesystem [3]). In the kernel versions 2.4 and earlier, the valid range of major numbers was 1-255 and 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 has since been lifted, and the 2.6 kernel should see an increase in the range of major and minor numbers available for use.

2 Problems with current scheme

2.1 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 in order 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 number 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 hotplug enabled systems, and hot-pluggable busses like IEEE1394, 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 is assigned to which physical device is now much easier to determine. For a system with two different USB printers plugged into it, the `sysfs /sys/class/usb` directory tree can look like Figure 1. 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 Figure 1, 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, by placing them both behind a USB hub, they might get renamed, as they are probed in a different order on startup.

In Figure 2, `/dev/usb/lp0` is assigned to the USB printer with the serial number W09090207101241330 due to this different probing order.

`sysfs` now enables a user to determine which device has been assigned by the kernel to which device file. This is a very powerful association that has not been previously 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 some configuration file somewhere, they just want to always be able to print to the proper printer, no matter where it is in the USB device tree.

```

/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

```

Figure 1: Two USB printers plugged into different USB busses

```

$ 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

```

Figure 2: Same USB printers plugged into a USB hub

2.2 Not enough numbers

The current range of allowed major and minor numbers is 8 bits (0-255). Currently there are very few major numbers left for new character devices, and about half the number of major numbers available for block devices (block and character devices can use the same numbers, but the kernel treats them separately, giving the whole range for both types of devices.)

This seems like a lot of free numbers, but there are users who want to use a very large number of disks all at the same time, for which the 8 bit scheme is too small. A common goal of some companies is to connect about 4,000 disks to a single system. For every disk, the kernel reserves 16 minor numbers, due to the possibility of there being up to 16 partitions on every disk. So 4,000 disks would require 64,000 different device files, needing at least 250 major num-

