

## udev and devfs – The final word

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-12/6377.html>

---

**From:** Greg KH ([greg\\_at\\_kroah.com](mailto:greg_at_kroah.com))

**Date:** 12/31/03

Date: Tue, 30 Dec 2003 16:29:42 -0800

To: [linux-hotplug-devel@lists.sourceforge.net](mailto:linux-hotplug-devel@lists.sourceforge.net), [linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org)

(This text can be found at [kernel.org/pub/linux/utils/kernel/hotplug/udev\\_vs\\_devfs](http://kernel.org/pub/linux/utils/kernel/hotplug/udev_vs_devfs) for those who want to link to it. I'll also update it with info based on the thread I know is going to spawn from this post...)

Executive summary for those too lazy to read this whole thing:

I don't care about devfs, and I don't want to talk about it at all anymore. If you love devfs, fine, I'm not trying to tell anyone what to do. But you really should be looking into using udev instead. All further email messages sent to me about devfs will be gladly ignored.

First off, some background. For a description of udev, and what it's original design goals were, please see the OLS 2003 paper on udev, available at:

[http://www.kroah.com/linux/talks/ols\\_2003\\_udev\\_paper/Reprint-Kroah-Hartman-OLS2003.pdf](http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf)

and the slides for the talk, available at:

[http://www.kroah.com/linux/talks/ols\\_2003\\_udev\\_talk/](http://www.kroah.com/linux/talks/ols_2003_udev_talk/)

The OLS paper can also be found in the docs/ directory of the udev tarball, available on kernel.org in the /pub/linux/utils/kernel/hotplug/ directory.

In that OLS paper, I described the current situation of a static /dev and the current problems that a number of people have with it. I also detailed how devfs tries to solve a number of these problems. In hindsight, I should have never mentioned the word, devfs, when talking about udev. I did so only because it seemed like a good place to start with. Most people understood what devfs is, and what it does. To compare udev against it, showing how udev was more powerful, and a more complete solution to the problems people were having, seemed like a natural comparison to me.

But no more. I hereby never want to compare devfs and udev again. With the exception of this message...

The Problems:

## Linux–Kernel: udev and devfs – The final word

- 1) A static /dev is unwieldy and big. It would be nice to only show the /dev entries for the devices we actually have running in the system.
- 2) We are (well, were) running out of major and minor numbers for devices.
- 3) Users want a way to name devices in a persistent fashion (i.e. "This disk here, must always be called "boot\_disk" no matter where in the scsi tree I put it", or "This USB camera must always be called "camera" no matter if I have other USB scsi devices plugged in or not.")
- 4) Userspace programs want to know when devices are created or removed, and what /dev entry is associated with them.

The constraints:

- 1) No policy in the kernel!
- 2) Follow standards (like the LSB)
- 3) must be small so embedded devices will use it.

So, how does devfs stack up to the above problems and constraints:

Problems:

- 1) devfs only shows the dev entries for the devices in the system.
- 2) devfs does not handle the need for dynamic major/minor numbers
- 3) devfs does not provide a way to name devices in a persistent fashion.
- 4) devfs does provide a daemon that userspace programs can hook into to listen to see what devices are being created or removed.

Constraints:

- 1) devfs forces the devfs naming policy into the kernel. If you don't like this naming scheme, tough.
- 2) devfs does not follow the LSB device naming standard.
- 3) devfs is small, and embedded devices use it. However it is implemented in non–pagable memory.

Oh yeah, and there are the insolvable race conditions with the devfs implementation in the kernel, but I'm not going to talk about them right now, sorry. See the linux–kernel archives if you care about them (and if you use devfs, you should care...)

So devfs is 2 for 7, ignoring the kernel races.

And now for udev:

Problems:

- 1) using udev, the /dev tree only is populated for the devices that are currently present in the system.
- 2) udev does not care about the major/minor number schemes. If the kernel tomorrow switches to randomly assign major and minor numbers to different devices, it would work just fine (this is exactly what I am proposing to do in 2.7...)
- 3) This is the main reason udev is around. It provides the ability to name devices in a persistent manner. More on that below.
- 4) udev emits D–BUS messages so that any other userspace program

(like HAL) can listen to see what devices are created or removed. It also allows userspace programs to query it's database to see what devices are present and what they are currently named as (providing a pointer into the sysfs tree for that specific device node.)

Constraints:

- 1) udev moves `_all_` naming policies out of the kernel and into userspace.
- 2) udev defaults to using the LSB device naming standard. If users want to deviate away from this standard (for example when naming some devices in a persistent manner), it is easily possible to do so.
- 3) udev is small (49Kb binary) and is entirely in userspace, which is swappable, and doesn't have to be running at all times.

Nice, 7 out of 7 for udev. Makes you think the problems and constraints were picked by a udev developer, right? No, the problems and constraints are ones I've seen over the years and so udev, along with the kernel driver model and sysfs, were created to solve these real problems. I also have had the luxury to see the problems that the current devfs implementation has, and have taken the time to work out something that does not have those same problems.

So by just looking at the above descriptions, everyone should instantly realize that udev is far better than devfs and start helping out udev development, right? Oh, you want more info, ok...

Back in May 2003 I released a very tiny version of udev that implemented everything that devfs currently does, in about 6Kb of userspace code:

<http://marc.theaimsgroup.com/?l=linux-kernel&m=105003185331553>

Yes, that's right, 6Kb. So, you are asking, why are you still working on udev if it did everything devfs did back in May 2003? That's because just managing static device nodes based on what the kernel calls the devices is `_not_` the primary goal of udev. It's just a tiny side affect of it's primary goal, the ability to never worry about major/minor number assignments and provide the ability to achieve persistent device names if wanted.

All the people wanting to bring up the udev vs. devfs argument go back and read the previous paragraph. Yes, all Gentoo users who keep filling up my inbox with smoking emails, I mean you.

So, how well does udev solve it's goals:

Prevent users from ever worrying about major/minor numbers  
And here you were, not knowing you ever needed to worry about major/minor numbers in the first place, right? Ah, I see you haven't plugged in 2 USB printers and tried to figure out which printer was which `/dev` entry? Or plugged in 4000 SCSI disks and tried to figure out how to access that 3642nd disk and what it was called in `/dev`. Or plugged in a USB camera and a USB flash drive

and then tried to download the pictures off of the flash drive by accident?

As the above scenarios show, both desktop users and big iron users both need to not worry about which device is assigned to what major/minor device.

udev doesn't care what major/minor number is assigned to a device. It merely takes the numbers that the kernel says it assigned to the device and creates a device node based on it, which the user can then use (if you don't understand the whole major/minor to device node issue, or even what a device node is, trust me, you don't really want to, go install udev and don't worry about it...) As stated above, if the kernel decides to start randomly assigning major numbers to all devices, then udev will still work just fine.

Provide a persistent device naming solution:

Lots of people want to assign a specific name that they can talk to a device to, no matter where it is in the system, or what order they plugged the device in. USB printers, SCSI disks, PCI sound cards, Firewire disks, USB mice, and lots of other devices all need to be assigned a name in a consistent manner (udev doesn't handle network devices, naming them is already a solved solution, using nameif). udev allows users to create simple rules to describe what device to name. If users want to call a program running a large database half–way around the world, asking it what to name this device, it can. We don't put the naming database into the kernel (like other Unix variants have), everything is in userspace, and easily accessible. You can even run a perl script to name your device if you are that crazy...

For more information on how to create udev rules to name devices, please see the udev man page, and look at the example udev rules that ship with the tarball.

So, convinced already why you should use udev instead of devfs? No. Ok, fine, I'm not forcing you to abandon your bloated, stifling policy, nonextensible, end of life feature if you don't want to. But please don't bother me about it either, I don't care about devfs, only about udev.

This is my last posting about this topic, all further emails sent to me about why devfs is wonderful, and why are you making fun of this wonderful, stable gift from the gods, will be gleefully ignored and possibly posted in a public place where others can see.

thanks,

greg k–h

–

## Linux-Kernel: udev and devfs – The final word

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to [majordomo@vger.kernel.org](mailto:majordomo@vger.kernel.org)  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>