

Modern Linux distributions are capable of identifying a hardware component which is plugged into an already-running system. There are a lot of user-friendly distributions like Ubuntu, which will automatically run specific applications like Rhythmbox when a portable device like an iPod is plugged into the system.

Hotplugging (which is the word used to describe the process of inserting devices into a running system) is achieved in a Linux distribution by a combination of three components: **Udev, HAL, and Dbus**.

Udev supplies a dynamic device directory containing only the nodes for devices which are connected to the system. It creates or removes the device node files in the /dev directory as they are plugged in or taken out. Dbus is like a system bus which is used for inter-process communication. The HAL gets information from the Udev service, when a device is attached to the system and it creates an XML representation of that device. It then notifies the corresponding desktop application like Nautilus through the Dbus and Nautilus will open the mounted device's files.

This article focuses only on **Udev**, which does the basic device identification.

What is Udev?

Udev is the device manager for the Linux 2.6 kernel that creates/removes device nodes in the /dev directory dynamically. It is the successor of devfs and hotplug. It runs in userspace and the user can change device names using Udev rules.

Udev depends on the sysfs file system which was introduced in the 2.5 kernel. It is sysfs which makes devices visible in user space. When a device is added or removed, kernel events are produced which will notify Udev in user space.

The external binary /sbin/hotplug was used in earlier releases to inform Udev about device state change. That has been replaced and Udev can now directly listen to those events through Netlink.

Why Do We Need It ?

In the older kernels, the /dev directory contained static device files. But with dynamic device creation, device nodes for only those devices which are actually present in the system are created. Let us see the disadvantages of the static /dev directory, which led to the development of Udev.

Problems Identifying the Exact Hardware Device for a Device Node in /dev

The kernel will assign a major/minor number pair when it detects a hardware device while booting the system. Let us consider two hard disks. The connection/alignment is in such a way that one is connected as a master and the other, as a slave. The Linux system will call them, */dev/hda*

and

/dev/hdb

. Now, if we interchange the disks the device name will change. This makes it difficult to identify the correct device that is related to the available static device node. The condition gets worse when there are a bunch of hard disks connected to the system.

Udev provides a persistent device naming system through the /dev directory, making it easier to identify the device.

The following is an example of persistent symbolic links created by Udev for the hard disks attached to a system.

```
$ ls -lR /dev/disk/  
/dev/disk/by-id:  
lrwxrwxrwx 1 root root 9 Jul 4 06:48 scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593  
-> ../../sda  
lrwxrwxrwx 1 root root 10 Jul 4 06:48  
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part1 -> ../../sda1  
lrwxrwxrwx 1 root root 10 Jul 4 06:48  
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part2 -> ../../sda2  
lrwxrwxrwx 1 root root 10 Jul 4 06:48  
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part3 -> ../../sda3  
lrwxrwxrwx 1 root root 10 Jul 4 06:48  
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part4 -> ../../sda4  
lrwxrwxrwx 1 root root 10 Jul 4 06:48  
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part5 -> ../../sda5
```

Written by unnikrishnan
Friday, 18 December 2009 10:14

```
lrwxrwxrwx 1 root root 10 Jul 4 06:48
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part6 -> ../../sda6
lrwxrwxrwx 1 root root 10 Jul 4 06:48
scsi-SATA_WDC_WD800JD-75M_WD-WMAM9UT48593-part7 -> ../../sda7
/dev/disk/by-label:
lrwxrwxrwx 1 root root 10 Jul 4 06:48 1 -> ../../sda6
lrwxrwxrwx 1 root root 10 Jul 4 06:48 boot1 -> ../../sda2
lrwxrwxrwx 1 root root 10 Jul 4 06:48 project -> ../../sda3
lrwxrwxrwx 1 root root 10 Jul 4 06:48 SWAP-sda7 -> ../../sda7
/dev/disk/by-path:
lrwxrwxrwx 1 root root 9 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0 -> ../../sda
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part1 -> ../../sda1
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part2 -> ../../sda2
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part3 -> ../../sda3
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part4 -> ../../sda4
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part5 -> ../../sda5
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part6 -> ../../sda6
lrwxrwxrwx 1 root root 10 Jul 4 06:48 pci-0000:00:1f.2-scsi-0:0:0:0-part7 -> ../../sda7
/dev/disk/by-uuid:
lrwxrwxrwx 1 root root 10 Jul 4 06:48 18283DC6283DA422 -> ../../sda1
lrwxrwxrwx 1 root root 10 Jul 4 06:48 25a4068c-e84a-44ac-85e6-461b064d08cd -> ../../sda6
lrwxrwxrwx 1 root root 10 Jul 4 06:48 3ea7cf15-511b-407a-a56b-c6bfa046fd9f -> ../../sda5
lrwxrwxrwx 1 root root 10 Jul 4 06:48 8878a0a4-604e-4ddf-b62c-637c4fa84d3f -> ../../sda2
lrwxrwxrwx 1 root root 10 Jul 4 06:48 e50bcd6d-61ea-4b05-81a8-3cbe17ad6674 -> ../../sda3
```

Persistent device naming helps to identify the hardware device without much trouble.

Huge Number of Device Nodes in /dev

In the static model of device node creation, no method was available to identify the hardware devices actually present in the system. So, device nodes were created for all the devices that Linux was known to support at the time. The huge mess of device nodes in /dev made it difficult to identify the devices actually present in the system.

Not Enough Major/Minor Number Pairs

The number of static device nodes to be included increased a lot in recent times and the 8-bit scheme, that was used, proved to be insufficient for handling all the devices. As a result the major/minor number pairs started running out.

Character devices and block devices have a fixed major/minor number pair assigned to them.

Written by unnikrishnan
Friday, 18 December 2009 10:14

The authority responsible for assigning the major/minor pair is the [Linux Assigned Name and Number Authority](#). But, a machine will not use all the available devices. So, there will be free major/minor numbers within a system. In such a situation, the kernel of that machine will borrow major/minor numbers from those free devices and assign those numbers to other devices which require it.

This can create issues at times. The user space application which handles the device through the device node will not be aware of the number change. For the user space application, the device number assigned by LANANA is very important. So, the user space application should be informed about the major/minor number change. This is called dynamic assignment of major/minor numbers and Udev does this task.

Udev's Goals

- Run in user space.
- Create persistent device names, take the device naming out of kernel space and implement rule based device naming.
- Create a dynamic /dev with device nodes for devices present in the system and allocate major/minor numbers dynamically.
- Provide a user space API to access the device information in the system.

Installation of Udev

Udev is the default device manager in the 2.6 kernel. Almost all modern Linux distributions come with Udev as part of the default installation. You can also get Udev from <http://www.kernel.org/pub/linux/utils/kernel/hotplug/>

. The latest version of Udev needs the 2.6.25 kernel with sysfs, procfs, signalfd, inotify, Unix domain sockets, networking, and hotplug enabled.

```
CONFIG_HOTPLUG=y
CONFIG_UEVENT_HELPER_PATH=""
CONFIG_NET=y
CONFIG_UNIX=y
CONFIG_SYSFS=y
CONFIG_SYSFS_DEPRECATED*=n
CONFIG_PROC_FS=y
CONFIG_TMPFS=y
CONFIG_TMPFS_POSIX_ACL=y
CONFIG_INOTIFY=y
CONFIG_SIGNALFD=y
```

For a much more reliable operation, the kernel must *not* use the

CONFIG_SYSFS_DEPRECATED* option.

Udev depends on the proc and sys file systems and they must be mounted on /proc and /sys.

Working of Udev

The Udev daemon listens to the netlink socket that the kernel uses for communicating with user space applications. The kernel will send a bunch of data through the netlink socket when a device is added to, or removed from a system. The Udev daemon catches all this data and will do the rest, i.e., device node creation, module loading etc.

Kernel Device Event Management

- When bootup is initialized, the /dev directory is mounted in tmpfs.
- After that, Udev will copy the static device nodes from `/lib/udev/devices` to the /dev directory.
- The Udev daemon then runs and collects uevents from the kernel, for all the devices connected to the system.
- The Udev daemon will parse the uevent data and it will match the data with the rules specified in `/etc/udev/rules.d`.
- It will create the device nodes and symbolic links for the devices as specified in the rules.
- The Udev daemon reads the rules from `/etc/udev/rules.d/*.rules` and stores them in the memory.
- Udev will receive an inotify event, if any rules were changed. It will read the changes and will update the memory.

Device Driver Loading For Devices

Udev uses the modalias method to load device drivers. The modalias file located at `/lib/modules/`uname -r`/modules.alias` helps Udev to load the drivers. The modalias file is created by the depmod binary and it contains alternate names for the device drivers.

Let us examine an example of device driver loading in Linux :

I am using a C program to collect data from the netlink socket that Udev uses to create device nodes and load modules.

```
[root@arch ~]# ./a.out
```

Udev: Introduction to Device Management In Modern Linux System

Written by unnikrishnan
Friday, 18 December 2009 10:14

```
add@/devices/pci0000:00/0000:00:02.1/usb1/1-4
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:02.1/usb1/1-4
SUBSYSTEM=usb
MAJOR=189
MINOR=1
DEVTYPE=usb_device
DEVICE=/proc/bus/usb/001/002
PRODUCT=1058/1010/105
TYPE=0/0/0
BUSNUM=001
DEVNUM=002
SEQNUM=1163
add@/devices/pci0000:00/0000:00:02.1/usb1/1-4/1-4:1.0
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:02.1/usb1/1-4/1-4:1.0
SUBSYSTEM=usb
DEVTYPE=usb_interface
DEVICE=/proc/bus/usb/001/002
PRODUCT=1058/1010/105 .....
```

You can see that it provides a lot of information about the device. This includes the modalias variable that tells Udev to load a particular module.

The modalias data will look like :

```
MODALIAS=pci:v000010ECd00008169sv00001385sd0000311Abc02sc00i00
```

The modalias data contains all the information required to find the corresponding device driver :

pci :- Its a pci device
v :- vendor ID of the device. Here it is 000010EC (ie 10EC)
d :- device ID of the device. Here it is 00008169 (ie : 8169)
sv and sd are subsystem versions for both vendor and device.

The best place to find the vendor/product from the id of a PCI device is <http://www.pcidatabase.com>.

Udev uses the modalias data to find the correct device driver from `/lib/modules/`uname -r`/modules.alias`.

```
$ grep -i 10EC /lib/modules/`uname -r`/modules.alias | grep -i 8169
```

Written by unnikrishnan
Friday, 18 December 2009 10:14

```
alias pci:v000010ECd00008129sv*sd*bc*sc*i* r8169  
alias pci:v000010ECd00008169sv*sd*bc*sc*i* r8169
```

You can see that the module which is suitable for the device is r8169. Let us get some more information about the driver.

```
$ /sbin/modinfo r8169  
filename: /lib/modules/2.6.18-53.el5/kernel/drivers/net/r8169.ko  
version: 2.2LK-NAPI  
license: GPL  
description: RealTek RTL-8169 Gigabit Ethernet driver  
author: Realtek and the Linux r8169 crew  
srcversion: D5EDA4980B92CA2CF677B62  
alias: pci:v00001737d00001032sv*sd00000024bc*sc*i*  
alias: pci:v000016ECd00000116sv*sd*bc*sc*i*  
alias: pci:v00001186d00004300sv*sd*bc*sc*i*  
alias: pci:v000010ECd00008129sv*sd*bc*sc*i*  
alias: pci:v000010ECd00008169sv*sd*bc*sc*i*  
depends:  
vermagic: 2.6.18-53.el5 SMP mod_unload 686 REGPARM 4KSTACKS gcc-4.1  
parm: media:force phy operation. Deprecated by ethtool (8). (array of int)  
parm: rx_copybreak:Copy breakpoint for copy-only-tiny-frames (int)  
parm: use_dac:Enable PCI DAC. Unsafe on 32 bit PCI slot. (int)  
parm: debug:Debug verbosity level (0=none, ..., 16=all) (int)
```

Check out the line starting with “depends”. It describes the other modules which the r8169 module depends on. Udev will load these modules also.

Rule Processing and Device Node Creation

As already mentioned, Udev parses the rules in `/etc/udev/rules.d/` for every device state change in the kernel. The Udev rule can be used to manipulate the device node name/permission/symlink in user space.

Let us see some sample rules that will help you understand Udev rules better.

The data supplied by the kernel through netlink is used by Udev to create the device nodes. The data includes the major/minor number pair and other device specific data such as device/vendor id, device serial number etc. The Udev rule can match all this data to change the name of the device node, create symbolic links or register the network link.

Udev: Introduction to Device Management In Modern Linux System

Written by unnikrishnan
Friday, 18 December 2009 10:14

The following example shows how to write a Udev rule to rename the network device in a system.

We need to get the device information to create a rule.

```
# udevadm info -a -p /sys/class/net/eth0/
```

looking at device '/devices/pci0000:00/0000:00:04.0/0000:01:06.0/net/eth0':

```
KERNEL=="eth0"  
SUBSYSTEM=="net"  
DRIVER=""  
ATTR{addr_len}=="6"  
ATTR{dev_id}=="0x0"  
ATTR{ifalias}=""  
ATTR{iflink}=="3"  
ATTR{ifindex}=="3"  
ATTR{features}=="0x829"  
ATTR{type}=="1"  
ATTR{link_mode}=="0"  
ATTR{address}=="00:80:48:62:2a:33"  
ATTR{broadcast}=="ff:ff:ff:ff:ff:ff"  
ATTR{carrier}=="1"  
ATTR{dormant}=="0"  
ATTR{operstate}=="unknown"  
ATTR{mtu}=="1500"  
ATTR{flags}=="0x1003"  
ATTR{tx_queue_len}=="1000"
```

looking at parent device '/devices/pci0000:00/0000:00:04.0/0000:01:06.0':

```
KERNELS=="0000:01:06.0"  
SUBSYSTEMS=="pci"  
DRIVERS=="8139too"  
ATTRS{vendor}=="0x10ec"  
ATTRS{device}=="0x8139"  
ATTRS{subsystem_vendor}=="0x10ec"  
ATTRS{subsystem_device}=="0x8139"  
ATTRS{class}=="0x020000"  
ATTRS{irq}=="19"  
ATTRS{local_cpus}=="ff"  
ATTRS{local_cpulist}=="0-7"  
ATTRS{modalias}=="pci:v000010ECd00008139sv000010ECsd00008139bc02sc00i00"  
ATTRS{enable}=="1"  
ATTRS{broken_parity_status}=="0"  
ATTRS{msi_bus}=""
```

looking at parent device '/devices/pci0000:00/0000:00:04.0':

Udev: Introduction to Device Management In Modern Linux System

Written by unnikrishnan
Friday, 18 December 2009 10:14

```
KERNELS=="0000:00:04.0"  
SUBSYSTEMS=="pci"  
DRIVERS=="  
ATTRS{vendor}=="0x10de"  
ATTRS{device}=="0x03f3"  
ATTRS{subsystem_vendor}=="0x0000"  
ATTRS{subsystem_device}=="0x0000"  
ATTRS{class}=="0x060401"  
ATTRS{irq}=="0"  
ATTRS{local_cpus}=="ff"  
ATTRS{local_cpulist}=="0-7"  
ATTRS{modalias}=="pci:v000010DEd000003F3sv00000000sd00000000bc06sc04i01"  
ATTRS{enable}=="1"  
ATTRS{broken_parity_status}=="0"  
ATTRS{msi_bus}=="1"  
looking at parent device '/devices/pci0000:00':  
KERNELS=="pci0000:00"  
SUBSYSTEMS=="  
DRIVERS=="
```

You can see that Udev has a lot of information about the network device. Let us examine it in detail :

```
KERNEL=="eth0" :- kernel name of the device is eth0  
DRIVERS=="8139too" :- driver loaded is 8139too  
ATTR{address}=="00:80:48:62:2a:33" :- hardware address of the device  
ATTRS{vendor}=="0x10ec" :- vendor id  
ATTRS{device}=="0x8139" :- device id
```

Let us create a rule to rename this network device to eth1 (This name will be persistent and will not be reset after a reboot).

```
>[root@arch ~]# cat /etc/udev/rules.d/70-persistent-net.rules  
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",  
ATTR{address}=="00:80:48:62:2a:33", ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
```

This rule renames the device to eth1. We can easily manage the network and other device nodes in the system, this way.

Udev Utilities

Udev provides some user space utilities to manage devices and device nodes in a system. One

Written by unnikrishnan
Friday, 18 December 2009 10:14

such command that you will find in all of the latest Linux distributions is 'udevadm'. The udevadm command is functionally capable of doing all the tasks which were done by the separate commands shown above.

This utility can be used to regenerate the device nodes in a running system as shown:

```
[root@arch ~]# ls -l /dev/ | wc -l
150
[root@arch ~]# rm -rf /dev/*
rm: cannot remove `/dev/pts/0': Operation not permitted
rm: cannot remove directory `/dev/shm': Device or resource busy
[root@arch ~]# ls -l /dev/ | wc -l
4
[root@arch ~]# udevadm trigger
[root@arch ~]# ls -l /dev/ | wc -l
150
```

There are many other useful operations that can be done using the udevadm command. You can get more information from the man page of udevadm.

What is the Future of Udev ?

It is impossible to predict the future of a Linux sub system. Linux is undergoing rapid development and it is probably not wise to predict the future of the Linux kernel. The DEVfs system which was introduced as a solution to static device nodes disappeared within a short span of time. But Udev has proven to be a successful device manager for the modern Linux kernel, and promises to be a more stable, feature rich device management system in future releases.

About the Author

Unnikrishnan A, is an expert in Linux Server Administration. Linux fascinates him as it is a platform for major opensource initiatives and gives any one, the ability to learn it deeply. He also admires the fact that Linux has a large community driven support which enables greater scope for emergence of ideas and solutions. Apart from Linux, Unnikrishnan also loves Electronics and Astronomy. During his free time, Unnikrishnan loves to explore the world of web designing. This article originally appeared at [Bobcares](#) .