Introduction to Linux dynamic device management

Birmingham Linux User Group
21 April 2011

Nick Morrott

Plan

- Device basics
- Intro to udev
- Writing udev rules

Part 1

Device basics

Linux device basics

- **All devices are files** (device nodes) e.g. /dev/sda1
- Abstraction allows device drivers/users to communicate with the 'real' devices as ordinary files
- Different types of device node (block, character, pseudo)

Device permissions

- Device nodes have **permissions** and **ownership**
- Just the same as 'regular' files

```
brw-rw---- 1 root disk 8, 1 Dec 11 12:00 /dev/sda1
```

Major/minor numbers

- Device nodes have **major/minor numbers** which identify the device driver (major) and specific device (minor) being controlled
- These are not present for 'regular' files

brw-rw---- 1 root disk 8, 1 Dec 11 12:00 /dev/sda1

Block devices (e.g. hard disk, optical media)

- Store and transmit data in structured sequences of bytes called **blocks**
- Use buffered I/O to boost performance, often support random access
- brw-rw---- 1 root disk 8, 1 Dec 11 12:00 /dev/sda1

  (major=8 is a block SCSI/SATA device, minor=1 is first partition on device)

Character devices (e.g. keyboards, terminals)

- Transmit data **single character at a time**
- I/O usually unbuffered, no support for random access seeking
- crw------- 1 root root 4, 1 Dec 11 12:00 /dev/tty1

  (major=4 is a char TTY device, minor=1 is the first virtual terminal)

“Pseudo” devices

- Not all device nodes on the system must correspond to physical devices:

  /dev/null
  - discards all input, produces no output

  /dev/zero
  - produces stream of NUL bytes

  /dev/urandom
  - produces stream of (pseudo)random numbers

Part 2

Introduction to udev

The ~~good~~ bad old days

Before Linux 2.5, device management was taken care of using **devfs**

devfs: the device filesystem

- Static list of devices created in /dev at installation time
- Nodes created for **all** possible devices (even if device was never installed)
- Implemented completely in the kernel
- No device-specific naming

Failings of devfs

- Static /dev was large and unwieldy
- Growing shortage of major/minor device numbers
- Real need for persistent device-specific naming
- Need for userspace notification when devices created/removed

  (from “udev and devfs – The Final Word”)

What happened?

devfs (and later hotplug and HAL) was deprecated and replaced in Linux 2.5 by the much more flexible **udev**

udev features

- Dynamic device nodes - only nodes for installed devices are created
- Implemented in userspace, allowing for:
  - notification of plug/unplug events
  - user to control device naming
  - querying of /sys to identify devices
- Support for persistent device naming - across reboots; with multiple similar devices; and with different hotplug ordering

sysfs: the system filesystem

- Virtual filesystem (/sys) present in Linux 2.6+
- Managed by kernel, browsable by user, can be queried with userspace tools
- Exports device information for installed hardware to userspace
- The device information is the magical ingredient that allows **udev** to create device nodes via **rules**

What tools does udev provide?

- **udevd** – user space daemon
- **libudev** – library providing access to device information
- **udevadm** – udev management/dianostics tool
- **udev rules** – match against the uevent and sysfs database to control device creation/naming

udevd

- Starts up in the background at boot and waits for **uevents**
- When a uevent is received it compares the information against udev's current set of rules for any matches
- As a bonus, new rules files are discovered automatically

udevadm

- Userspace tool to manage/query/test udev
- Replaces udevinfo (to which older tutorials may still refer)
- 'udevadm --info' is used to query udev database for a given device
- 'udevadm --test' is used to test a udev event run for a given device

udevadm info example (1) Intel SSD

```
$ /sbin/udevadm info --query=property –name=/dev/sda

UDEV_LOG=3
DEVPATH=/devices/pci0000:00/0000:00:08.0/host2/target2:0:0/2:0:0:0/block/sd
a
MAJOR=8
MINOR=0
DEVNAME=/dev/sda
DEVTYPE=disk
SUBSYSTEM=block
ID_ATA=1
ID_TYPE=disk
ID_BUS=ata
ID_MODEL=INTEL_SSDSA2M040G2GC
ID_MODEL_ENC=INTEL\x20SSDSA2M040G2GC\x20\x20\x20\x20\x20\x20\x20
\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
ID_REVISION=2CV102DH
```

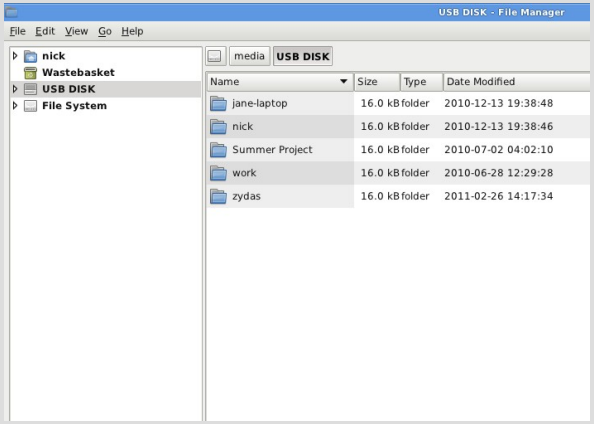
udevadm info example (2) Logitech USB Headset

```
$ /sbin/udevadm info --query=property --name=/dev/snd/by-id/usb-Logitech_Logitech_USB_Headset-00

UDEV_LOG=3
DEVPATH=/devices/pci0000:00/0000:00:02.0/usb2/2-7/2-7:1.0/sound/card3/controlC3
MAJOR=116
MINOR=21
DEVNAME=/dev/snd/controlC3
SUBSYSTEM=sound
ID_VENDOR=Logitech
ID_VENDOR_ENC=Logitech
ID_VENDOR_ID=046d
ID_MODEL=Logitech_USB_Headset
ID_MODEL_ENC=Logitech\x20USB\x20Headset
ID_MODEL_ID=0a02
ID_REVISION=1013
ID_SERIAL=Logitech_Logitech_USB_Headset
ID_TYPE=audio
```

That's all well and good, but

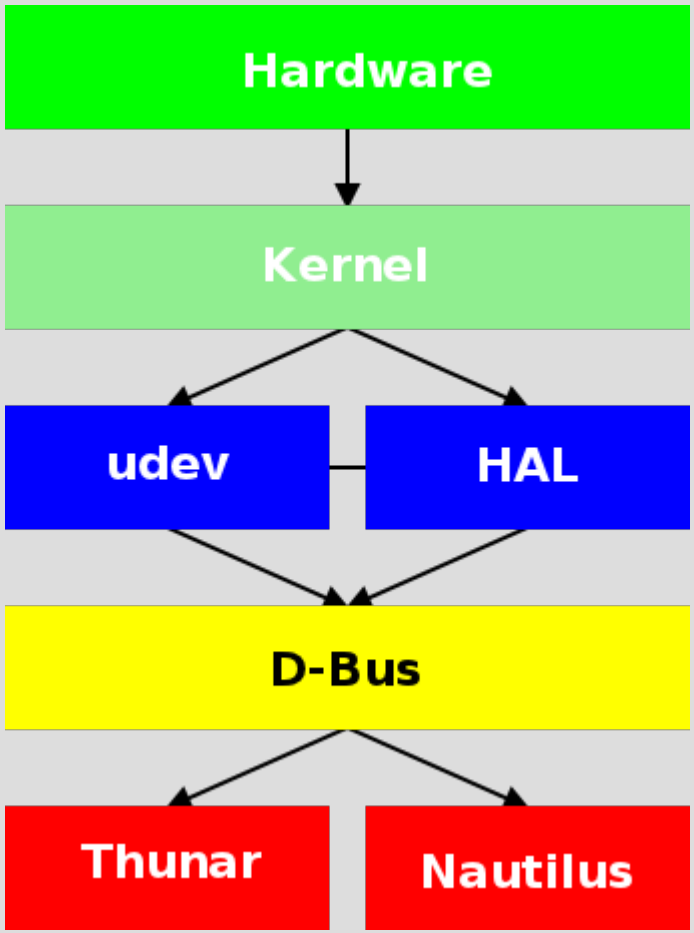
How does $FILE_MANAGER open a new file browser when you plug in your memory stick?



udev, HAL, D-Bus

- Kernel recognises new hardware, loads relevant modules and triggers uevent
- udev/sysfs responsible for creating the device node(s)
- udev messages the device info on D-Bus
- A D-Bus-registered $FILE_MANAGER receives the information, and opens up a new file browser

From hardware to browser



(Adapted from Linux Magazine #71, 10/2006)

Current status

As always, Linux libraries and tools are in a state of flux

- HAL was deprecated in merged into udev
- DeviceKit (HAL's modular replacement) was itself deprecated and also rolled into udev, UPower and udisks
- Newer kernels (2.6.32+) can use devtmpfs with/without udev

Part 3

Writing udev rules

udev rules (1)

- determine how devices get created (name, permissions, ownership)
- udev comes with a set of default rules (in /lib/udev/rules.d/), but you can write your own (stick them in /etc/udev/rules.d/10-local.rules)
- Rules files must have the extension **.rules** and are parsed in lexical order
- Rules must be on a single line

udev rules (2)

- >1 rule can match a particular device
- all matches will be processed unless a rule states no further processing should take place (OPTIONS+="last_rule")
- udev creates one 'real' node for a particular device, but multiple symlinks can be created for more flexibility
- Rules can match against a multitude of exported device information

udev rules (3)

- rules use **key-value pairs** to match a particular device (based on uevent and sysfs information)
- Multiple key-value pairs allow for more granular device control
- Keys (**match** or **assignment**) are used to select a particular property
- Values are used to specify a property's value
- Operators (==, +=, =) link keys to values

Simple udev rule example (1)

Scenario

We have a SATA drive (sdb) backups that we want configured with a persistent name to use with backup scripts

Solution

Match the device named 'sdb' by the kernel, and create a device node '/dev/backup_disk'

KERNEL=="sdb", NAME="backup_disk"

Simple udev rule example (1)

Scenario

We have a SATA drive (sdb) backups that we want configured with a persistent name to use with backup scripts

Solution

Match the device named 'sdb' by the kernel, and create a device node '/dev/backup_disk'

KERNEL=="sdb", NAME="backup_disk"

Simple udev rule example (1)

Scenario

We have a SATA drive (sdb) backups that we want configured with a persistent name to use with backup scripts

Solution

Match the device named 'sdb' by the kernel, and create a device node '/dev/backup_disk'

KERNEL=="sdb", NAME="backup_disk"

Simple udev rule example (2)

Scenario

Instead of naming the 'real' device node /dev/backup_disk, we want the device named /dev/sdb (regular kernel name) but additionally create a symlink to the device called /dev/backup_disk

Solution

KERNEL=="sdb", SYMLINK+="backup_disk"

Simple udev rule example (2)

Scenario

Instead of naming the 'real' device node /dev/backup_disk, we want the device named /dev/sdb (regular kernel name) but additionally create a symlink to the device called /dev/backup_disk

Solution

KERNEL=="sdb", SYMLINK+="backup_disk"

Simple udev rule example (2)

Scenario

Instead of naming the 'real' device node /dev/backup_disk, we want the device named /dev/sdb (regular kernel name) but additionally create a symlink to the device called /dev/backup_disk

Solution

KERNEL=="sdb", SYMLINK+="backup_disk"

Using sysfs attributes

- We can use `udevadm info` to list all exported attributes for a given device
- We are not limited to using only sysfs attributes: we can mix kernel, driver, subsystem and sysfs match keys as required
- So, back to the udevadm output for the Logitech USB headset for a moment...

sysfs udev rule example (1) Logitech USB Headset

```
$ /sbin/udevadm info --query=property --name=/dev/snd/by-id/usb-Logitech_Logitech_USB_Headset-00

UDEV_LOG=3
DEVPATH=/devices/pci0000:00/0000:00:02.0/usb2/2-7/2-7:1.0/sound/card3/controlC3
MAJOR=116
MINOR=21
DEVNAME=/dev/snd/controlC3
SUBSYSTEM=sound
ID_VENDOR=Logitech
ID_VENDOR_ENC=Logitech
ID_VENDOR_ID=046d
ID_MODEL=Logitech_USB_Headset
ID_MODEL_ENC=Logitech\x20USB\x20Headset
ID_MODEL_ID=0a02
ID_REVISION=1013
ID_SERIAL=Logitech_Logitech_USB_Headset
ID_TYPE=audio
```

sysfs udev rule example (1) Logitech USB Headset

Scenario

We want to match the headset (the only one we have) when it is connected/disconnected

Solution

Looking at the exposed sysfs attributes, it would appear that using the 'ID_MODEL' attribute should be unique enough for us

(We could also use the 'SUBSYSTEM' attribute to further restrict the match)

sysfs udev rule example (1) Logitech USB Headset

```
$ /sbin/udevadm info --query=property --name=/dev/snd/by-id/usb-Logitech_Logitech_USB_Headset-00

UDEV_LOG=3
DEVPATH=/devices/pci0000:00/0000:00:02.0/usb2/2-7/2-7:1.0/sound/card3/controlC3
MAJOR=116
MINOR=21
DEVNAME=/dev/snd/controlC3
SUBSYSTEM=sound
ID_VENDOR=Logitech
ID_VENDOR_ENC=Logitech
ID_VENDOR_ID=046d
ID_MODEL=Logitech_USB_Headset
ID_MODEL_ENC=Logitech\x20USB\x20Headset
ID_MODEL_ID=0a02
ID_REVISION=1013
ID_SERIAL=Logitech_Logitech_USB_Headset
ID_TYPE=audio
```

sysfs udev rule example (2) Logitech USB Headset

- Let's construct the match component of the rule using these two attributes:

  SUBSYSTEM==”sound”, ATTR{ID_MODEL}==”Logitech_USB_Headset”

- Now that we can identify the headset, instead of changing the name of the device node created when it is plugged in, let's make the rule call an external script to set it as the default ALSA device

sysfs udev rule example (2) Logitech USB Headset

- Let's construct the match component of the rule using these two attributes:

  SUBSYSTEM==”sound”, ATTR{ID_MODEL}==”Logitech_USB_Headset”

- Now that we can identify the headset, instead of changing the name of the device node created when it is plugged in, let's make up the rule that might call an external script to set it as the default ALSA device

sysfs udev rule example (3) Logitech USB Headset

- If we want to run a script/program to give us a device name to use in a rule, we use the PROGRAM assignment key
- If we just want to run an external script or program when a rule is triggered, we use the RUN assignment key
- We can further improve the rule by using the ACTION match key to determine whether the device is being connected or disconnected from the system

sysfs udev rule example (4) Logitech USB Headset

- Here's a set of rules that might meet our requirements:

```
# first match rule uses ID_MODEL
ENV{ID_MODEL}=="Logitech_USB_Headset", GOTO="logitech_start"

# shortcut if we haven't matched the headset
GOTO="logitech_end"

# we've matched, so we run these rules
LABEL="logitech_start"
ACTION=="add", RUN+="/usr/local/bin/headset-connect.sh"
ACTION=="remove", RUN+="/usr/local/bin/headset-disconnect.sh"

LABEL="logitech_end"

# (adapted from http://ubuntuforums.org/showthread.php?t=559014)
```

Testing before deployment

- It would be great if we could test the effect of rules before we deploy them – we can!
- Increase debug verbosity:

  udevadm control --log-priority="debug-verbose"
- and check the effect of the current rules:

  udevadm test devpath

  (where devpath is an absolute sysfs path rooted on /sys)

(Finally) udev rules for MythTV (1)

Scenario

We have several devices (DVB/video/LIRC) in a system but their device nodes (/dev/videon and /dev/dvb/adaptern) move around on boot - we want persistent device nodes on **every** boot

Solution

udev rules!

udev rules for MythTV (2)

- We have DVB-T (Freeview) and DVB-S (Freesat) cards in this example:
- We want to always refer to the DVB-S card as /dev/dvb/adapter101/

```
# Create a symlink /dev/dvb/adapter101 pointing to Nova S Plus
# with bus ID 0000:03:04.2
SUBSYSTEM=="dvb", ATTRS{vendor}=="0x14f1", \
KERNELS=="0000:03:04.2", \
PROGRAM="/bin/sh -c 'K=%k; K=$${K#dvb}; printf dvb/adapter101/%
%s $${K#*.}'", \
SYMLINK+="%c"
```

udev rules for MythTV (3)

- We have DVB-T and DVB-S cards in this example:
- We want to always refer to the DVB-T card as /dev/dvb/adapter102/

```
# Create a symlink /dev/dvb/adapter102 pointing to K-World 210
# with bus ID 0000:03:05.0
SUBSYSTEM=="dvb", ATTRS{vendor}=="0x1131", \
KERNELS=="0000:03:05.0", \
PROGRAM="/bin/sh -c 'K=%k; K=$${K#dvb}; printf dvb/adapter102/%
%s $${K#*.}'", \
SYMLINK+="%c"
```

udev rules for MythTV (4)

Scenario

Our IR remote is correctly detected but changes node number on reboot which breaks LIRC

Solution

Create a persistent node for LIRC to use

```
# Create a symlink /dev/input/irremote pointing to the IR port
# on the Nova S Plus
KERNEL=="event[0-9]", \
ATTRS{name}=="cx88 IR (Hauppauge Nova-S-Plus*", \
SYMLINK+="input/irremote"
```

Useful reading/links

- udev homepage
  (http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html)
- udev and devfs – The Final Word
  (http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev_vs_devfs)
- Making Hardware Just Work
  (http://www.ometer.com/hardware.html)
- Linux Allocated Device List
  (http://www.kernel.org/pub/linux/docs/device-list/devices.txt)
- Writing udev rules
  (http://www.reactivated.net/writing_udev_rules.html)

Thanks for listening!

Questions?