Linux Device Management: Getting to know udev

Ryan Matteson
matty91@gmail.com
http://prefetch.net

Overview

- Tonight I am going to discuss udev, and show how this amazing technology can be used to handle all of your device management needs
- I am planning to split my presentation into two parts:
  - Part 1 will provide an overview of udev
  - Part 2 will show how to use udev to configure a USB storage device that is hot plugged into my laptop

What is udev?

- Udev is a device management framework that replaced the devfs facility in the Linux 2.6 kernel
- Provides a number of features:
  - Dynamic creation of nodes in /dev
  - Persistent naming of devices (has anyone had sda become sdb after a reboot?)
  - Provides a flexible rule engine that can be used to control every facet (device name, owner, group, permissions, etc.) of the device creation process
  - Allows arbitrary programs to be run when devices are added and removed from a system

How does udev work?

- When the kernel detects that a device has been added or removed, a uevent is sent to the udevd daemon through a netlink socket
- When udevd receives the uevent, it matches its configured rules against the available device attributes provided in sysfs
- If a match is found, one or more actions (e.g., create device node, remove device node, install firmware, etc.) are taken

Udev rules

- Udev rules are added to files in /etc/udev/rules.d, and take the following form:

  MATCH_KEY(S) ASSIGNMENT(S)

- MATCH_KEY takes the form of one or more key/value match statements, and the ASSIGNMENT equates to one or more actions to perform when a match occurs

Udev rules (cont.)

- Match keys can include a number of items:
  - Kernel subsystem the device is part of (KERNEL)
  - Driver type (DRIVER)
  - One or more sysfs attributes (ATTRS)
  - Numerous more (udev(7) lists them all) …
- Regular expressions (*, ?, [0-9], etc.) can be used as part of the match expression, and match results are provided to the rule in the form of one or more % variables

Udev rules (cont.)

- To get a listing of sysfs attributes for a given device, you can run the ***udevinfo*** utility:

```
$ udevinfo -a -p /block/sdb
KERNELS=="1-2"
SUBSYSTEMS=="usb"
DRIVERS=="usb"
ATTRS{serial}=="35A3FB10074017271004”
```

- These attributes can be used as part of your matching criteria

Udev rules (cont.)

- After a match is made, one or more actions can be invoked:
  - Set the device name (NAME)
  - Create a symbolic link to the device (SYMLINK)
  - Change the owner of the device (OWNER)
  - Change the permissions of the device (MODE)
  - Numerous more (udev(7) lists them all) …
- The PROGRAM action allows you to run arbitrary programs during rule processing, and the output from this program is available for matching via the RESULT key

Udev example

- Example: Say we have a USB key drive that we want to mount at /dev/usbdrive1
- We can add a rule similar to the following to /etc/udev/rules.d/s10-usbdrive.rules:

```
SUBSYSTEM=="block", SUBSYSTEMS=="usb", \
ATTRS{serial}=="35A3FB10074017271004",  \
NAME="usbdrive1", OWNER="matty", \
MODE="0600"
```

How does this rule work?

- Using the previous example, the rule will apply the following logic:
  - Is the device in the block subsystem?
  - Is the device a child of the usb driver?
  - Does the device serial number (as acquired from the sysfs file system) equal 35A3FB10074017271004?
- If the three rules match, a device node named /dev/usbdisk1 with an owner of matty and the permission 0600 will be created in /dev

Testing udev rules

- Once you create a new udev rule, you can use the ***udevtest*** utility to verify that your rule works:

```
$ udevtest /block/sdb
main: looking at device '/block/sdb' from subsystem 'block'
udev_rules_apply_to_event: OWNER 501 \
/etc/udev/rules.d/  S10-usbdrive.rules:1
udev_rules_apply_to_event: MODE 0600 \
/etc/udev/rules.d/S10-usbdrive.rules:1
udev_rules_apply_to_event: NAME 'usbdisk1' \
/etc/udev/rules.d/S10-usbdrive.rules:1
```

- The program takes the sysfs device node as an argument, and prints the rules that would be applied to the device

Monitoring udev activity

- You can observe the uevents passed between the kernel and udevd with the ***udevmonitor*** utility:

```
$ udevmonitor
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent
KERNEL[1253142121.510109] add      \
            /devices/pci0000:00/0000:00:02.1/usb1/1-2 (usb)
KERNEL[1253142121.511227] add     \
           /devices/pci0000:00/0000:00:02.1/usb1/1-2/1-2:1.0 (usb)
UDEV  [1253142121.521639] add     \
           /devices/pci0000:00/0000:00:02.1/usb1/1-2 (usb)
UDEV  [1253142121.528646] add      \
          /devices/pci0000:00/0000:00:02.1/usb1/1-2/usb_endpoint/    \
          usbdev1.2_ep00 (usb_endpoint)
```

Debugging your udev rules

- If your devices aren’t being created, there is a three step process you can use to find out why:
  - Run udevtest to see howyour rule is interpreted
  - Use udevmonitor to observe the uevents
  - Enable debug logging with udevcontrol
- If that fails, you can ‘yumdownloader –source udev’ and dig through the callout programs and udevd source to see how a given rule should be handled (this is a last resort, as the steps above should identify the issue)

Conclusion

- The udev device management framework is extremely powerful, and allows you to control every facet of how device nodes are created
- To learn about some of udev's advanced capabilities (e.g., using regular expressions, running custom scripts, etc.), please check out the udev(7) manual page and the documentation in the kernel source code

Questions?

References

- Persistent device naming in userspace
  http://www.linuxjournal.com/article/7316
- Devfs vs. udev write up:
  http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev_vs_devfs
- Udev(7) manual page:
  http://linux.die.net/man/7/udev
- Writing udev rules:
  http://www.reactivated.net/writing_udev_rules.html