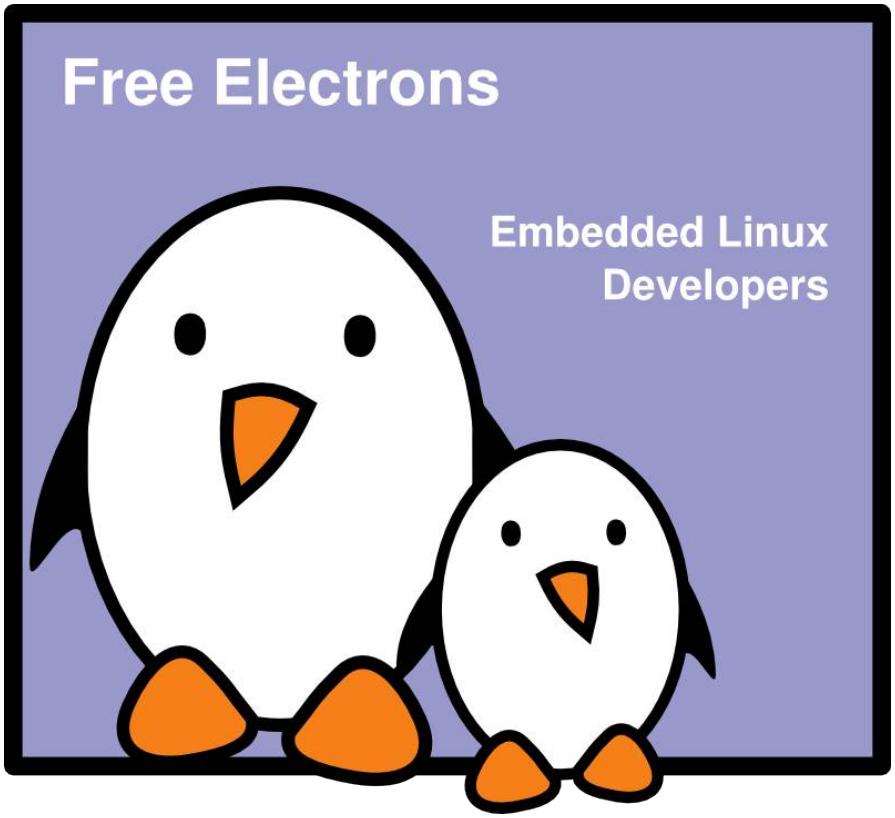
Hotplugging with udev

Michael Opdenacker
Free Electrons



Latest update: Sep 28, 2010,
Document sources, updates and translations:
http://free-electrons.com/docs/udev
Corrections, suggestions, contributions and translations are welcome!

/dev issues and limitations

- On Red Hat 9, 18000 entries in `/dev`!
  All entries for all possible devices
  had to be created at system installation.
- Needed an authority to assign major numbers
  http://lanana.org/: Linux Assigned Names and Numbers Authority
- Not enough numbers in 2.4, limits extended in 2.6.
- Userspace neither knew what devices were present in the system, nor which real device corresponded to each `/dev` entry.

The udev solution

Takes advantage of sysfs introduced by Linux 2.6.

- Created by Greg Kroah Hartman, a huge contributor.
  Other key contributors: Kay Sievers, Dan Stekloff.
- Entirely in user space.
- Automatically creates / removes device entries
  in `/dev/` according to inserted / removed devices.
- Major and minor device transmitted by the kernel.
- Requires no change to driver code.
- Fast: written in C
  Small size: `udevd` version 108: 61 KB in Ubuntu 7.04

Starting udev (1)

- At the very beginning of user-space startup, mount the `/dev/` directory as a tmpfs filesystem:
  `sudo mount -t tmpfs udev /dev`
- `/dev/` is populated with static devices available in `/lib/udev/devices/` :

Ubuntu 6.10 example:

```
crw------- 1 root root   5, 1 2007-01-31 04:18 console
lrwxrwxrwx 1 root root     11 2007-01-31 04:18 core -> /proc/kcore
lrwxrwxrwx 1 root root     13 2007-01-31 04:18 fd -> /proc/self/fd
crw-r----- 1 root kmem   1, 2 2007-01-31 04:18 kmem
brw------- 1 root root   7, 0 2007-01-31 04:18 loop0
lrwxrwxrwx 1 root root     13 2007-01-31 04:18 MAKEDEV -> /sbin/MAKEDEV
drwxr-xr-x 2 root root   4096 2007-01-31 04:18 net
crw------- 1 root root   1, 3 2007-01-31 04:18 null
crw------- 1 root root 108, 0 2007-01-31 04:18 ppp
drwxr-xr-x 2 root root   4096 2006-10-16 14:39 pts
drwxr-xr-x 2 root root   4096 2006-10-16 14:39 shm
lrwxrwxrwx 1 root root     24 2007-01-31 04:18 sndstat -> /proc/asound/oss/sndstat
lrwxrwxrwx 1 root root     15 2007-01-31 04:18 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root     15 2007-01-31 04:18 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root     15 2007-01-31 04:18 stdout -> /proc/self/fd/1
```

Starting udev (2)

- The `udevd` daemon is started.
  It listens to *uevents* from the driver core,
  which are sent whenever devices are inserted or removed.
- The `udevd` daemon reads and parses all the rules found in
  `/etc/udev/rules.d/`
  and keeps them in memory.
- Whenever rules are added, removed or modified,
  `udevd` receives an *inotify* event and updates its
  ruleset in memory.
- When an event is received, `udevd` starts a process to:
  - try to match the event against udev rules,
  - create / remove device files,
  - and run programs (to load / remove a driver, to notify user space...)

The ***inotify*** mechanism lets userspace programs subscribe to notifications of filesystem changes. Possibility to watch individual files or directories.

uevent message example

Example inserting a USB mouse

```
recv(4,                                          // socket id
     "add@/class/input/input9/mouse2\0           // message
     ACTION=add\0                                // action type
     DEVPATH=/class/input/input9/mouse2\0        // path in /sys
     SUBSYSTEM=input\0                           // subsystem (class)
     SEQNUM=1064\0                               // sequence number
     PHYSDEVPATH=/devices/pci0000:00/0000:00:1d.1/usb2/2-2/2-2:1.0\0
                                                 // device path in /sys
     PHYSDEVBUS=usb\0                            // bus
     PHYSDEVDRIVER=usbhid\0                      // driver
     MAJOR=13\0                                  // major number
     MINOR=34\0",                                // minor number
     2048,                                       // message buffer size
     0)                                          // flags
= 221                                            // actual message size
```

udev rules

When a udev rule matching event information is found,
it can be used:

- To define the name and path of a device file.
- To define the owner, group and permissions of a device file.
- To execute a specified program.

Rule files are processed in lexical order.

udev naming capabilities

Device names can be defined

- from a label or serial number,
- from a bus device number,
- from a location on the bus topology,
- from a kernel name,
- from the output of a program.

See http://www.reactivated.net/writing_udev_rules.html for a very complete description. See also `man udev`.

udev naming rule examples

```
# Naming testing the output of a program
BUS=="scsi", PROGRAM="/sbin/scsi_id", RESULT=="OEM 0815", NAME="disk1"

# USB printer to be called lp_color
BUS=="usb", SYSFS{serial}=="W09090207101241330", NAME="lp_color"

# SCSI disk with a specific vendor and model number will be called boot
BUS=="scsi", SYSFS{vendor}=="IBM", SYSFS{model}=="ST336", NAME="boot%n"

# sound card with PCI bus id 00:0b.0 to be called dsp
BUS=="pci", ID=="00:0b.0", NAME="dsp"

# USB mouse at third port of the second hub to be called mouse1
BUS=="usb", PLACE=="2.3", NAME="mouse1"

# ttyUSB1 should always be called pda with two additional symlinks
KERNEL=="ttyUSB1", NAME="pda", SYMLINK="palmtop handheld"

# multiple USB webcams with symlinks to be called webcam0, webcam1, ...
BUS=="usb", SYSFS{model}=="XV3", NAME="video%n", SYMLINK="webcam%n"
```

udev permission rule examples

Excerpts from `/etc/udev/rules.d/40-permissions.rules`

```
# Block devices
SUBSYSTEM!="block", GOTO="block_end"
SYSFS{removable}!="1",                    GROUP="disk"
SYSFS{removable}=="1",                    GROUP="floppy"
BUS=="usb",                               GROUP="plugdev"
BUS=="ieee1394",                          GROUP="plugdev"
LABEL="block_end"

# Other devices, by name

KERNEL=="null",                           MODE="0666"
KERNEL=="zero",                           MODE="0666"
KERNEL=="full",                           MODE="0666"
```

Identifying device driver modules

Kernel / module compiling

Each driver announces which device and vendor ids it supports. Information stored in module files.

↓

The `depmod -a` command processes module files and generates `/lib/modules/<version>/modules.alias`

System everyday life

The driver core (usb, pci...) reads the device id, vendor id and other device attributes.

↓

The kernel sends an event to `udevd`, setting the `MODALIAS` environment variable, encoding these data.

↓

A udev event process runs `modprobe $MODALIAS`

↓

`modprobe` finds the module to load in the `modules.alias` file.

Module aliases

- `MODALIAS` environment variable example (USB mouse):
  `MODALIAS=usb:v046DpC03Ed2000dc00dsc00dp00ic03isc01ip02`
- Matching line in `/lib/modules/<version>/modules.alias`:
  `alias usb:v*p*d*dc*dsc*dp*ic03isc01ip02* usbmouse`

udev modprobe rule examples

Even module loading is done with udev!

Excerpts from `/etc/udev/rules.d/90-modprobe.rules`

```
ACTION!="add", GOTO="modprobe_end"


SUBSYSTEM!="ide", GOTO="ide_end"
IMPORT{program}="ide_media --export $devpath"
ENV{IDE_MEDIA}=="cdrom",  RUN+="/sbin/modprobe -Qba ide-cd"
ENV{IDE_MEDIA}=="disk",   RUN+="/sbin/modprobe -Qba ide-disk"
ENV{IDE_MEDIA}=="floppy", RUN+="/sbin/modprobe -Qba ide-floppy"
ENV{IDE_MEDIA}=="tape",   RUN+="/sbin/modprobe -Qba ide-tape"
LABEL="ide_end"

SUBSYSTEM=="input", PROGRAM="/sbin/grepmap --udev", \
           RUN+="/sbin/modprobe -Qba $result"

# Load drivers that match kernel-supplied alias

ENV{MODALIAS}=="?*", RUN+="/sbin/modprobe -Q $env{MODALIAS}"
```

Coldplugging

- Issue: loosing all device events happening during kernel initialization, because udev is not ready yet.
- Solution: after starting `udevd`, have the kernel emit uevents for all devices present in `/sys`.
- This can be done by the `udevtrigger` utility.
- Strong benefit: completely transparent for userspace.
  Legacy and removable devices handled and named in exactly the same way.

Debugging events - udevmonitor (1)

`udevadm monitor` visualizes the driver core events
and the `udev` event processes.
Example event sequence connecting a USB mouse:

```
UEVENT[1170452995.094476] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UEVENT[1170452995.094569] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UEVENT[1170452995.098337] add@/class/input/input28
UEVENT[1170452995.098618] add@/class/input/input28/mouse2
UEVENT[1170452995.098868] add@/class/input/input28/event4
UEVENT[1170452995.099110] add@/class/input/input28/ts2
UEVENT[1170452995.099353] add@/class/usb_device/usbdev4.30
UDEV  [1170452995.165185] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UDEV  [1170452995.274128] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV  [1170452995.375726] add@/class/usb_device/usbdev4.30
UDEV  [1170452995.415638] add@/class/input/input28
UDEV  [1170452995.504164] add@/class/input/input28/mouse2
UDEV  [1170452995.525087] add@/class/input/input28/event4
UDEV  [1170452995.568758] add@/class/input/input28/ts2
```

It gives time information measured in microseconds.
You can measure time elapsed between the uevent (UEVENT line), and the completion of the corresponding udev process (matching UDEV line).

Debugging events - udevmonitor (2)

`udevadm monitor --env`

shows the complete event environment for each line.

```
UDEV  [1170453642.595297] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV_LOG=3
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
SUBSYSTEM=usb
SEQNUM=3417
PHYSDEVBUS=usb
DEVICE=/proc/bus/usb/004/031
PRODUCT=46d/c03d/2000
TYPE=0/0/0
INTERFACE=3/1/2
MODALIAS=usb:v046DpC03Dd2000dc00dsc00dp00ic03isc01ip02
UDEVD_EVENT=1
```

Misc udev utilities

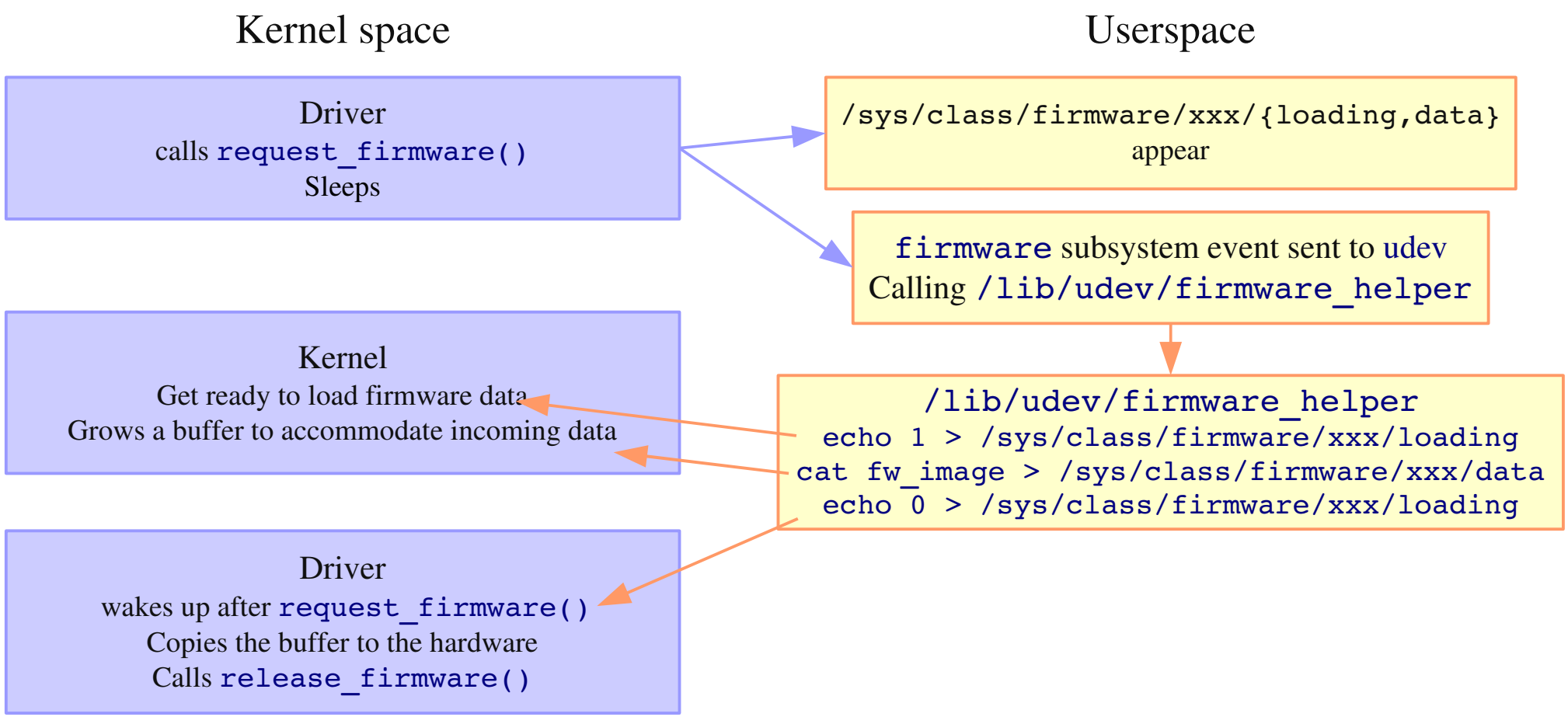
- `udevinfo`
  Lets users query the udev database.
- `udevtest <sysfs_device_path>`
  Simulates a `udev` run to test the configured rules.

Firmware hotplugging

Also implemented with udev!

- Firmware data are kept outside device drivers
  - May not be legal or free enough to distribute
  - Firmware in kernel code would occupy memory permanently, even if just used once.
- Kernel configuration: needs to be set in `CONFIG_FW_LOADER` (Device Drivers -> Generic Driver Options -> hotplug firmware loading support)

Firmware hotplugging implementation



See `Documentation/firmware_class/` for a nice overview

udev files

- `/etc/udev/udev.conf`
  udev configuration file.
  Mainly used to configure syslog reporting priorities.
  Example setting: `udev_log="err"`
- `/lib/udev/rules.d/`
  Standard udev event matching rules, installed by the distribution.
- `/etc/udev/rules.d/*.rules`
  Local (custom) udev event matching rules. Best to modify these.
- `/lib/udev/devices/*`
  static `/dev` content (such as `/dev/console`, `/dev/null`...).
- `/lib/udev/*`
  helper programs called from udev rules.
- `/dev/*`
  Created device files.

Kernel configuration for udev

Created for 2.6.19

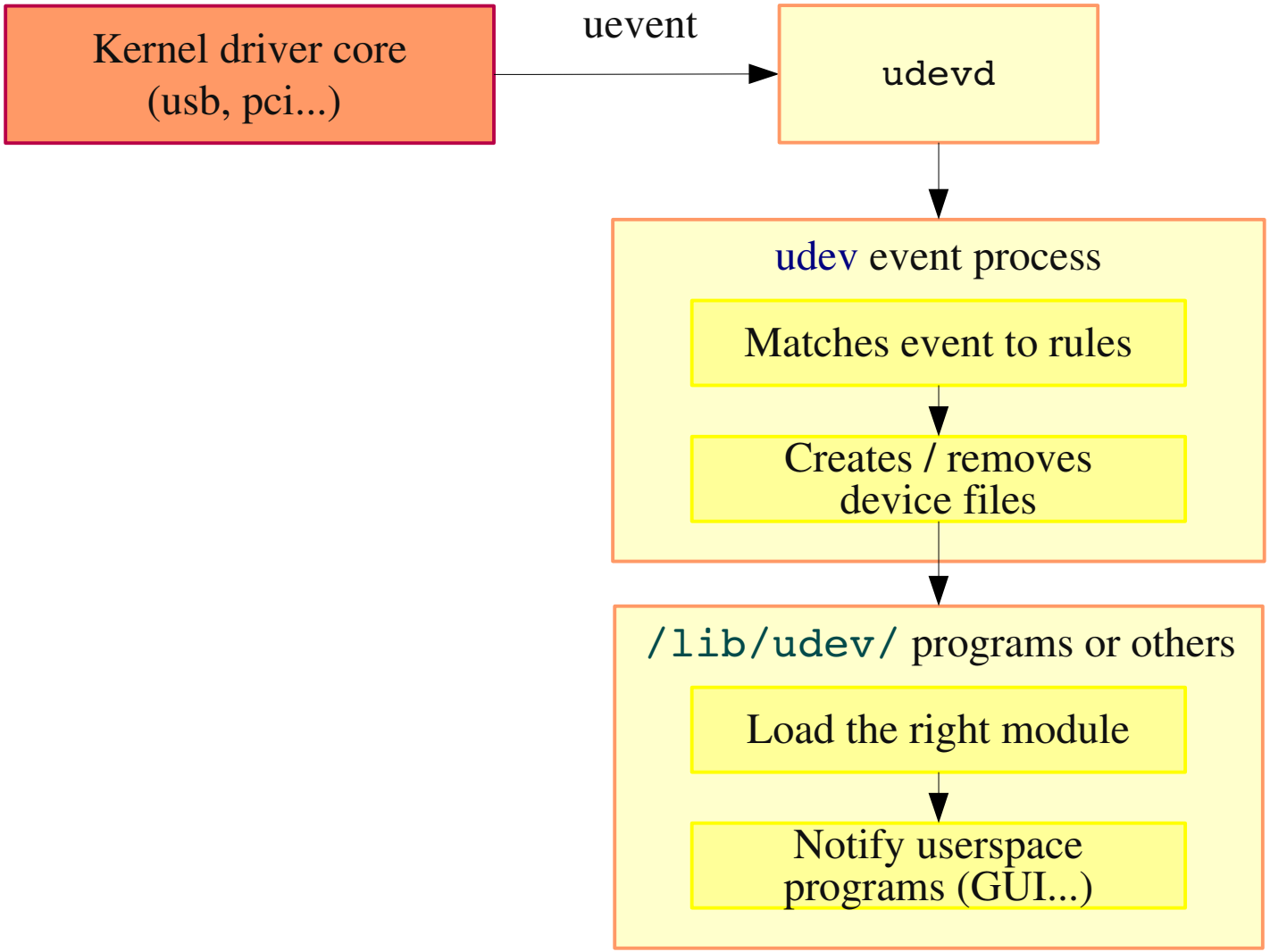
Caution: no documentation found, and not tested yet on a minimalistic system.
Some settings may still be missing.
Subsystems and device drivers (USB, PCI, PCMCIA...) should be added too!

```
# General setup
CONFIG_HOTPLUG=y
# Networking, networking options
CONFIG_NET=y
CONFIG_UNIX=y
CONFIG_NETFILTER_NETLINK=y
CONFIG_NETFILTER_NETLINK_QUEUE=y
# Pseudo filesystems
CONFIG_PROC_FS=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_RAMFS=y
```

Unix domain sockets

Needed to manage `/dev`

udev summary - typical operation



udev resources

- Home page
  http://kernel.org/pub/linux/utils/kernel/hotplug/udev.html
- Sources
  http://kernel.org/pub/linux/utils/kernel/hotplug/
- The udev manual page:
  `man udev`

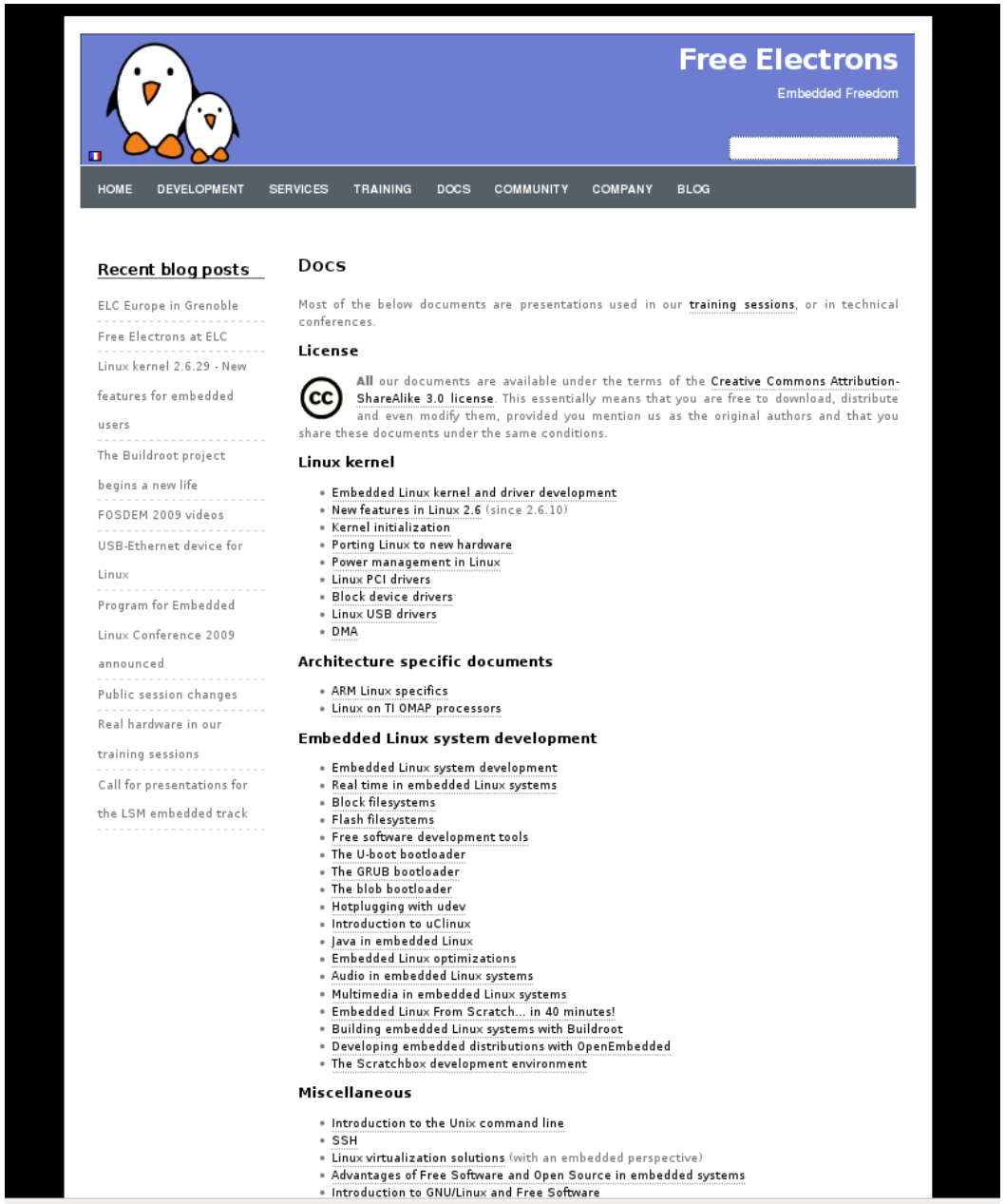
mdev, the udev for embedded systems

- udev might be too heavy-weight for some embedded systems, the `udevd` daemon staying in the background waiting for events.
- BusyBox provides a simpler alternative called mdev, available by enabling the `MDEV` configuration option.
- mdev's usage is documented in `doc/mdev.txt` in the BusyBox source code.
- mdev is also able to load firmware to the kernel like udev

mdev usage

- To use mdev, the proc and sysfs filesystems must be mounted
- *mdev* must be enabled as the hotplug event manager
  `echo /sbin/mdev > /proc/sys/kernel/hotplug`
- Need to mount `/dev` as a tmpfs:
  `mount -t tmpfs mdev /dev`
- Tell mdev to create the `/dev` entries corresponding to the devices detected during boot when mdev was not running:
  `mdev -s`
- The behavior is specified by the `/etc/mdev.conf` configuration file, with the following format
  `<device regex> <uid>:<gid> <octal permissions> [=path] [@|$|*<command>]`
- Example
  `hd[a-z][0-9]* 0:3 660`

Related documents



Free Electrons
Embedded Freedom

HOME DEVELOPMENT SERVICES TRAINING DOCS COMMUNITY COMPANY BLOG

Recent blog posts

- ELC Europe in Grenoble
- Free Electrons at ELC
- Linux kernel 2.6.29 - New features for embedded users
- The Buildroot project begins a new life
- FOSDEM 2009 videos
- USB-Ethernet device for Linux
- Program for Embedded Linux Conference 2009 announced
- Public session changes
- Real hardware in our training sessions
- Call for presentations for the LSM embedded track

Docs

Most of the below documents are presentations used in our training sessions, or in technical conferences.

License

All our documents are available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license. This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

Linux kernel

- Embedded Linux kernel and driver development
- New features in Linux 2.6 (since 2.6.10)
- Kernel initialization
- Porting Linux to new hardware
- Power management in Linux
- Linux PCI drivers
- Block device drivers
- Linux USB drivers
- DMA

Architecture specific documents

- ARM Linux specifics
- Linux on TI OMAP processors

Embedded Linux system development

- Embedded Linux system development
- Real time in embedded Linux systems
- Block filesystems
- Flash filesystems
- Free software development tools
- The U-boot bootloader
- The GRUB bootloader
- The blob bootloader
- Hotplugging with udev
- Introduction to uClinux
- Java in embedded Linux
- Embedded Linux optimizations
- Audio in embedded Linux systems
- Multimedia in embedded Linux systems
- Embedded Linux From Scratch... in 40 minutes!
- Building embedded Linux systems with Buildroot
- Developing embedded distributions with OpenEmbedded
- The Scratchbox development environment

Miscellaneous

- Introduction to the Unix command line
- SSH
- Linux virtualization solutions (with an embedded perspective)
- Advantages of Free Software and Open Source in embedded systems
- Introduction to GNU/Linux and Free Software

All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development

How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs