Linux USB device drivers
Training lab book

Michael Opdenacker
Free Electrons
http://free-electrons.com

About this document

This document is part of an embedded Linux training from Free Electrons.

You will find the whole training materials (slides and lab book) on http://free-electrons.com/training/devtools.

Lab data can be found on http://free-electrons.com/labs/embedded_linux.tar.bz2.

Copying this document

Document updates and translations available on http://free-electrons.com/training/devtools

Corrections, suggestions, contributions and translations are welcome!

Training setup

See the training labs on http://free-electrons.com/training/drivers for setup instructions, which are shared with these practical labs.

Lab 1 - Basic USB device driver

Objective: Get familiar with the basic kernel API to control USB devices.

After this lab, you will

- be a bit more familiar with automatic loading of USB modules with udev.
- have a bit of practice with USB driver registration.

Setup

Go to the `/mnt/labs/usb/lab1/` directory.

Pick up a USB device (webcam, sound card...). It is preferred not to choose mouse or keyboard input devices, as interacting with them may cause trouble in typing commands or using a graphical interface.

Device identification

Insert your device.

Check your kernel log and see messages corresponding to this insertion.

Using the `usbview` interface, find the vendor and product ids for your device.

If you don't have `usbview`, download its package. In Debian (logged as `root`):

```
apt-get update
apt-get install usbview
```

Deactivating automatic loading of a default driver

Unless it is fairly recent, your device is likely to already have a working driver in your GNU/Linux distribution for your current kernel. This existing driver will interfere with the new driver that you will try to create, so will will disable it on your system:

- Identify this driver from the `dmesg` output and from the list of loaded modules (insert and remove the device again if needed).
- Go to the modules directory for the running kernel:
  `cd /lib/modules/`uname -r``
- Backup the `modules.usbmap` file, used by `udev` to find out which module to load when a device is inserted:
  `cp modules.usbmap modules.usbmap.orig`
- Before making changes, remove the device.
- Find the line in `modules.usbmap` matching your module name, VendorId and ProductId, and remove it.
- Check that the default driver no longer gets loaded when you insert the device.

Removing the device causes the default driver module to be unloaded, hopefully for the last time.

Driver registration

Remove the device.

You will insert the device again when your new driver is ready to test.

Reusing the `usblab.c` file in your lab directory, create a Linux kernel module which registers a driver for this device, at least `probe()` and `disconnect()` functions.

You will test this new module directly on your development PC, compiling the module using the kernel headers available in the KernelKit distribution.

Using `printk()` messages, make sure your `probe()` and `disconnect()` callbacks get called when you insert and remove the device.

Accessing device information

Improve the probe() function to display with `printk()`, for each endpoint in the current interface:

- The direction of the endpoint (`IN` or `OUT`).
- The type of the endpoint (control, interrupt, bulk or isochronous).

A very limited lab

Unfortunately, end of the lab so far.

Implementing actual communication with the device would require deeper knowledge about how to interact with the device (or with a device of its class, according to the USB specifications), what to expect from it...

Another strong requirement is to be familiar with the subsystem from which user interaction is going to come, deciding to open the device, read from it, write to it. You would have to register your device with the input subsystem for keyboard and mice, with the video4linux subsystem for webcams, with the sound system for sound cards... That's extra learning that you would need before starting to use your device (learn from code examples from drivers for similar devices!).

However, you may have a try with the endpoints that you've just listed, and at least try to get control messages from the device...

Or you may try to mimic an existing driver for a similar device...