

Writing USB Device Drivers

Greg Kroah-Hartman <greg@kroah.com>

Writing USB Device Drivers

by Greg Kroah-Hartman

Copyright © 2001-2002 Greg Kroah-Hartman

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

This documentation is based on an article published in Linux Journal Magazine, October 2001, Issue 90.

Table of Contents

1. Introduction	1
2. Linux USB Basics	2
3. Device operation	4
4. Isochronous Data	7
5. Conclusion	8
6. Resources	9

Chapter 1. Introduction

The Linux USB subsystem has grown from supporting only two different types of devices in the 2.2.7 kernel (mice and keyboards), to over 20 different types of devices in the 2.4 kernel. Linux currently supports almost all USB class devices (standard types of devices like keyboards, mice, modems, printers and speakers) and an ever-growing number of vendor-specific devices (such as USB to serial converters, digital cameras, Ethernet devices and MP3 players). For a full list of the different USB devices currently supported, see Resources.

The remaining kinds of USB devices that do not have support on Linux are almost all vendor-specific devices. Each vendor decides to implement a custom protocol to talk to their device, so a custom driver usually needs to be created. Some vendors are open with their USB protocols and help with the creation of Linux drivers, while others do not publish them, and developers are forced to reverse-engineer. See Resources for some links to handy reverse-engineering tools.

Because each different protocol causes a new driver to be created, I have written a generic USB driver skeleton, modelled after the `pci-skeleton.c` file in the kernel source tree upon which many PCI network drivers have been based. This USB skeleton can be found at `drivers/usb/usb-skeleton.c` in the kernel source tree. In this article I will walk through the basics of the skeleton driver, explaining the different pieces and what needs to be done to customize it to your specific device.


```
module_init(usb_skel_init);
```

When the driver is unloaded from the system, it needs to deregister itself with the USB subsystem. This is done with the `usb_deregister` function:

```
static void __exit usb_skel_exit(void)
{
    /* deregister this driver with the USB subsystem */
    usb_deregister(&skel_driver);
}
module_exit(usb_skel_exit);
```

To enable the linux-hotplug system to load the driver automatically when the device is plugged in, you need to create a `MODULE_DEVICE_TABLE`. The following code tells the hotplug scripts that this module supports a single device with a specific vendor and product ID:

```
/* table of devices that work with this driver */
static struct usb_device_id skel_table [] = {
    { USB_DEVICE(USB_SKEL_VENDOR_ID, USB_SKEL_PRODUCT_ID) },
    { } /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, skel_table);
```

There are other macros that can be used in describing a `usb_device_id` for drivers that support a whole class of USB drivers. See `usb.h` for more information on this.

One of the more difficult problems that USB drivers must be able to handle smoothly is the fact that the USB device may be removed from the system at any point in time, even if a program is currently talking to it. It needs to be able to shut down any current reads and writes and notify the user-space programs that the device is no longer there. The following code (function `skel_delete`) is an example of how to do this:

```
static inline void skel_delete (struct usb_skel *dev)
{
    kfree (dev->bulk_in_buffer);
    if (dev->bulk_out_buffer != NULL)
        usb_free_coherent (dev->udev, dev->bulk_out_size,
                           dev->bulk_out_buffer,
                           dev->write_urb->transfer_dma);
    usb_free_urb (dev->write_urb);
    kfree (dev);
}
```

If a program currently has an open handle to the device, we reset the flag `device_present`. For every read, write, release and other functions that expect a device to be present, the driver first checks this flag to see if the device is still present. If not, it releases that the device has disappeared, and a `-ENODEV` error is returned to the user-space program. When the release function is eventually called, it determines if there is no device and if not, it does the cleanup that the `skel_disconnect` function normally does if there are no open files on the device (see Listing 5).

Chapter 4. Isochronous Data

This usb-skeleton driver does not have any examples of interrupt or isochronous data being sent to or from the device. Interrupt data is sent almost exactly as bulk data is, with a few minor exceptions. Isochronous data works differently with continuous streams of data being sent to or from the device. The audio and video camera drivers are very good examples of drivers that handle isochronous data and will be useful if you also need to do this.

Chapter 5. Conclusion

Writing Linux USB device drivers is not a difficult task as the usb-skeleton driver shows. This driver, combined with the other current USB drivers, should provide enough examples to help a beginning author create a working driver in a minimal amount of time. The linux-usb-devel mailing list archives also contain a lot of helpful information.

Chapter 6. Resources

The Linux USB Project: <http://www.linux-usb.org/> [<http://www.linux-usb.org>]

Linux Hotplug Project: <http://linux-hotplug.sourceforge.net/> [<http://linux-hotplug.sourceforge.net>]

Linux USB Working Devices List: <http://www.qbik.ch/usb/devices/> [<http://www.qbik.ch/usb/devices>]

linux-usb-devel Mailing List Archives: <http://marc.theaimsgroup.com/?l=linux-usb-devel>

Programming Guide for Linux USB Device Drivers: <http://usb.cs.tum.edu/usbdoc>

USB Home Page: <http://www.usb.org>