Write a real Linux driver

Greg Kroah-Hartman
SuSE Labs / Novell
gregkh@suse.de
greg@kroah.com

Agenda

- Intro to kernel modules
- sysfs basics
- USB basics
- Driver to device binding

Agenda - continued

- Using sysfs
- Sending and receiving USB data

Hello world kernel module

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>

static int __init gotemp_init(void)
{
	printk(KERN_INFO "Hello from the kernel!\n");
	return 0;
}

static void __exit gotemp_exit(void)
{
}

module_init(gotemp_init);
module_exit(gotemp_exit);

MODULE_AUTHOR("My name here");
MODULE_DESCRIPTION("Simple driver");
MODULE_LICENSE("GPL");
```

Makefile

```
obj-m := hello.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD       := $(shell pwd)

all:
	$(MAKE) -C $(KERNELDIR) M=$(PWD)
```

Manipulating Modules

- See the module informaion

  ```
  modinfo hello.ko
  ```

- Insert the module into the kernel

  ```
  insmod hello.ko
  ```

- See that it did something

  ```
  lsmod
  dmesg
  ```

- Remove the module from the kernel

  ```
  rmmod hello
  ```

sysfs crash course

"Web woven by a spider on drugs"

- lwn.net

/proc is for processes, not for drivers

USB crash course

- devices and configs and interfaces oh my

- How do you identify a unique device?

  MODULE_DEVICE_TABLE
  - "it just works"

Bind to the device - step 1

```
#define VENDOR_ID   0x08f7
#define PRODUCT_ID  0x0002

/* table of devices that work with this driver */
static struct usb_device_id id_table [] = {
        { USB_DEVICE(VENDOR_ID, PRODUCT_ID) },
        { },
};
MODULE_DEVICE_TABLE(usb, id_table);
```

Bind to the device - step 2

```
static int gotemp_probe(struct usb_interface *interface,
                        const struct usb_device_id *id)
{
        dev_info(&interface->dev,
                 "USB GoTemp device now attached\n");
        return 0;
}

static void gotemp_disconnect(struct usb_interface *interface)
{
        dev_info(&interface->dev,
                 "USB GoTemp now disconnected\n");
}
```

Bind to the device - step 3

```
static struct usb_driver gotemp_driver = {
        .name =       "gotemp",
        .probe =      gotemp_probe,
        .disconnect = gotemp_disconnect,
        .id_table =   id_table,
};

static int __init gotemp_init(void)
{
        return usb_register(&gotemp_driver);
}

static void __exit gotemp_exit(void)
{
        usb_deregister(&gotemp_driver);
}
```

Device specific structure

```
struct gotemp {
	struct usb_device *udev;
	int temp;
};
```

Device specific structure

```
static int gotemp_probe(struct usb_interface *interface,
                        const struct usb_device_id *id)
{
        struct usb_device *udev = interface_to_usbdev(interface);
        struct gotemp *gdev;

        gdev = kmalloc(sizeof(struct gotemp), GFP_KERNEL);
        if (gdev == NULL) {
                dev_err(&interface->dev, "Out of memory\n");
                return -ENOMEM;
        }
        memset(gdev, 0x00, sizeof(*gdev));

        gdev->udev = usb_get_dev(udev);

        usb_set_intfdata(interface, gdev);

        dev_info(&interface->dev, "USB GoTemp device now attached\n");
        return 0;
}
```

Device specific structure

```
static void gotemp_disconnect(struct usb_interface *interface)
{
        struct gotemp *gdev;

        gdev = usb_get_intfdata(interface);
        usb_set_intfdata(interface, NULL);

        usb_put_dev(gdev->udev);

        kfree(gdev);

        dev_info(&interface->dev, "USB GoTemp now disconnected\n");
}
```

Make a sysfs file

```
static ssize_t show_temp(struct device *dev,
                        struct device_attribute *attr,
                        char *buf)
{
        struct usb_interface *intf = to_usb_interface(dev);
        struct gotemp *gdev = usb_get_intfdata(intf);

        return sprintf(buf, "%d\n", gdev->temp);
}
static DEVICE_ATTR(temp, S_IRUGO, show_temp, NULL);
```

- In gotemp_probe:

```
device_create_file(&interface->dev, &dev_attr_temp);
```

- In gotemp_disconnect:

```
device_remove_file(&interface->dev, &dev_attr_temp);
```

Simple USB data transfer

```
#define CMD_ID_START_MEASUREMENTS 0x18
#define CMD_ID_INIT               0x1A

struct output_packet {
        u8 cmd;
        u8 params[7];
} __attribute__((packed));

static int send_cmd(struct gotemp *gdev, u8 cmd)
{
        struct output_packet *pkt;
        int retval;

        pkt = kmalloc(sizeof(*pkt), GFP_KERNEL);
        if (!pkt)
                return -ENOMEM;
        memset(pkt, 0x00, sizeof(*pkt));
        pkt->cmd = cmd;
```

Simple USB data transfer

send_cmd() continued:

```
        retval = usb_control_msg(gdev->udev,
                                usb_sndctrlpipe(gdev->udev, 0),
                                0x09,    /* bRequest = SET_REPORT */
                                0x21,    /* bRequestType = 00100001 */
                                0x0200,  /* or is it 0x0002? */
                                0x0000,  /* interface 0 */
                                pkt, sizeof(*pkt), 10000);
        if (retval == sizeof(*pkt))
                retval = 0;
        kfree(pkt);
        return retval;
}

static void init_dev(struct gotemp *gdev)
{
        /* First send an init message */
        send_cmd(gdev, CMD_ID_INIT);

        /* Start sending measurements */
        send_cmd(gdev, CMD_ID_START_MEASUREMENTS);
}
```

"Real" USB data transfers

struct urb;

- dynamically created
- fire and forget
- high data rates
- Like network skb packets

urb in the device structure

```
struct gotemp {
        struct usb_device *udev;
        int temp;
        unsigned char *int_in_buffer;
        struct urb *int_in_urb;
};
```

Find the endpoint

```
gotemp_probe():

        /* find the one control endpoint of this device */
        endpoint = &interface->cur_altsetting->endpoint[0].desc;

        buffer_size = le16_to_cpu(endpoint->wMaxPacketSize);

        gdev->int_in_buffer = kmalloc(buffer_size, GFP_KERNEL);
        if (!gdev->int_in_buffer) {
                dev_err(&interface->dev,
                        "Could not allocate buffer\n");
                goto error;
        }
```

Create the urb

gotemp_probe():

```
        gdev->int_in_urb = usb_alloc_urb(0, GFP_KERNEL);
        if (!gdev->int_in_urb) {
                dev_err(&interface->dev, "No free urbs available\n");
                goto error;
        }
        usb_fill_int_urb(gdev->int_in_urb, udev,
                        usb_rcvintpipe(udev,
                                endpoint->bEndpointAddress),
                        gdev->int_in_buffer, buffer_size,
                        read_int_callback, gdev,
                        endpoint->bInterval);
```

Submit the urb

```
init_dev():

        /* kick off interrupt urb */
        retval = usb_submit_urb(gdev->int_in_urb, GFP_KERNEL);
        if (retval)
                dev_err(&gdev->udev->dev,
                        "Error %d submitting interrupt urb\n",
                        retval);
```

The data from the device

```
struct measurement_packet {
        u8    measurements_in_packet;
        u8    rolling_counter;
        le16  measurement0;
        le16  measurement1;
        le16  measurement2;
} __attribute__((packed));
```

The urb callback

```
static void read_int_callback(struct urb *urb, struct pt_regs *regs)
{
        struct gotemp *gdev = urb->context;
        struct measurement_packet *measurement = urb->transfer_buffer;
        int retval;

        switch (urb->status) {
        case 0: break;
        case -ECONNRESET:
        case -ENOENT:
        case -ESHUTDOWN:
                /* this urb is terminated, clean up */
                dbg("urb shutting down with status: %d",
                    urb->status);
                return;
        default:
                dbg("nonzero urb status received: %d",
                    __FUNCTION__, urb->status);
                goto exit;
        }
```

The urb callback

read_int_callback() continued:

```
        gdev->temp = le16_to_cpu(measurement->measurement0);

exit:
        retval = usb_submit_urb(urb, GFP_ATOMIC);
        if (retval)
                dev_err(&urb->dev->dev,
                        "Error %d submitting interrupt urb\n",
                        retval);
}
```

The End