1 Design Process & Implementation

Design

In writing the USB Missile Launcher driver, the main goals were to figure out how the missile launcher was being communicated to by the computer, write the basic outline for a usb driver, as well as figure out the best way to write a userland program to interact with the launcher.

First, we needed to to "sniff" out to interaction with the missile launcher in a Windows environment. We booted into windows and sniffed the usage of the proprietary driver's communication process, discovering that the communication was done by sending a different number for each type of control: left, right, up, down, fire, and stop.

The last thing that needed to be done was to write the userland control software for the launcher, needing to chose which language and whether or not we continuously ran the program until and exit sequence or making the program a "run once" type of application.

the biggest design choice that helped us in the project was to try to understand how a basic usb drivers works, such as registering with teh system and how it communicates with the computer. Once we had that down, we made the driver a module which drastically cut down the time wasted waiting for a kernel to compile, installing the kernel, and rebooting the kernel.

Implementation

In implementing the encrypted ramdisk driver, our group first needed to understand the different functions required for the block device driver to be usable for the kernel. In order to encrypt or decrypt the data from the device, the group needed to understand specifically the encryption and decryption of block and bytes.

In the implementation of the missile launcher driver, we created a basic usb driver outline to make sure that the usb device was seen by the system. Once we had the basic outline completed, we booted a computer into a windows install and sniffed out how the the launcher communicates with the computer and it's control sequences.

This turned out to be very simplistic in that only a single number is sent at a time for each type of control: up, down, left, right, fire, and stop, and that each number stayed in to the motor controller until it was updated with a new value.

After we had communication with the launcher down, we had to figure out how to setup a type of interrupt that stopped the launcher from moving once it reached the limit in its range of movement, otherwise damage could be done to the launcher.

Due to the way the missile launcher was seen by the system, we had to combat the HID driver from taking over control of the missile launcher. this meant we had to unbind the device from the HID driver, and rebind it with the launcher driver. This seemed like the easiest way in which to do this at the time, and not go and patch the HID driver.

As for the userland control application, due to the limitations of the squiddly (no X11 and control over a serial interface), we decided to use a "run once" c program with a configurable timer to interact with the launcher. If we had an X11 environment, we would have used python the the Tkinter library which had alibrary that would allow events to be triggered on a key press and a key release.

Testing

In testing our best-fit implementation, our group created two seperate kernels: one with the first fit slob, and 1 with the best-fit slob. After booting up and running some programs to thrash the memory, we compared the output from /proc/buddyinfo to compare the memory usage and fragmentation.

In testing our usb driver, we compared the visual movement of the launcher with that of the Windows driver and the integrated driver in later Linux kernels for comparison.

We also tested the accuracy of the control between the two environments, and they seemed to at least match up across the different software environments.

Some weird results we noticed while trying to thrash our system memory, we noticed that malloc() didn't have as much effect on the memory fragemnetation as we expected. This is due to the behavior of malloc, so we switched to using sbrk() (which malloc uses) to allocate larger sections of memory. This resulted in a more noticeable memory effect. We also notice that there was not as much memory difference when comparing the mbuddyinfo before and buddyinfo after out memory thrashing program. But there was a difference after compiling a program.

Issues

Once issue we faced was trying to determine the name of the device to unbind from the HID driver, since they are listed with weird numbers instead of by a common name.

2 Code

```
--- /home/geoff/school/linux/drivers/usb/ml.c        1969-12-31 16:00:00.000000000 -0800
+++ kernel/drivers/usb/ml.c        2013-06-06 15:29:36.000000000 -0700
@@ -0,0 +1,649 @@
+/*
+ * Dream Cheeky USB Thunder Launcher driver
+ *
+ * Copyright (C) 2012 Nick Glynn <Nick.Glynn@feabhas.com>
+ *
+ * This program 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, version 2.
+ *
+ * derived from the USB Missile Launcher driver
+ * Copyright (C) 2007 Matthias Vallentin <vallentin@icsi.berkeley.edu>
+ *
+ * Note: The device node "/dev/launcher0" is created with root:root so you
+ *       will need to chgrp, chmod or sudo your way to access
+ */
+
+#include <linux/module.h>
+#include <linux/kernel.h>
+#include <linux/usb.h>
+#include <linux/slab.h>
+#include <linux/usb.h>
+#include <linux/mutex.h>
+#include <linux/ioctl.h>
```

```
+#include <asm/uaccess.h>
+
+/* Rocket launcher specifics */
+#define LAUNCHER_VENDOR_ID              0x2123
+#define LAUNCHER_PRODUCT_ID             0x1010
+
+#define LAUNCHER_NODE                   "launcher"
+#define LAUNCHER_CTRL_BUFFER_SIZE       8
+#define LAUNCHER_CTRL_REQUEST_TYPE      0x21
+#define LAUNCHER_CTRL_REQUEST           0x09
+#define LAUNCHER_CTRL_VALUE             0x0
+#define LAUNCHER_CTRL_INDEX             0x0
+#define LAUNCHER_CTRL_COMMAND_PREFIX    0x02
+
+#define LAUNCHER_STOP                   0x20
+#define LAUNCHER_UP                     0x02
+#define LAUNCHER_DOWN                   0x01
+#define LAUNCHER_LEFT                   0x04
+#define LAUNCHER_RIGHT                  0x08
+#define LAUNCHER_UP_LEFT                (LAUNCHER_UP | LAUNCHER_LEFT)
+#define LAUNCHER_DOWN_LEFT              (LAUNCHER_DOWN | LAUNCHER_LEFT)
+#define LAUNCHER_UP_RIGHT               (LAUNCHER_UP | LAUNCHER_RIGHT)
+#define LAUNCHER_DOWN_RIGHT             (LAUNCHER_DOWN | LAUNCHER_RIGHT)
+#define LAUNCHER_FIRE                   0x10
+
+#define LAUNCHER_MAX_UP                 0x80    /* 80 00 00 00 00 00 00 00 */
+#define LAUNCHER_MAX_DOWN               0x40    /* 40 00 00 00 00 00 00 00 */
+#define LAUNCHER_MAX_LEFT               0x04    /* 00 04 00 00 00 00 00 00 */
+#define LAUNCHER_MAX_RIGHT              0x08    /* 00 08 00 00 00 00 00 00 */
+
+static struct usb_class_driver class;
+static DEFINE_MUTEX(disconnect_mutex);
+
+struct usb_ml {
+        struct usb_device               *udev;
+        struct usb_interface            *interface;
+        unsigned char                   minor;
+        char                            serial_number[8];
+
+        /* Open count for this port */
+        int                             open_count;
+
+        /* Locks this structure */
+        struct                          semaphore sem;
+
+        /* locks dev->command */
+        spinlock_t                      cmd_spinlock;
+
+        char                            *int_in_buffer;
+        struct usb_endpoint_descriptor  *int_in_endpoint;
+        struct urb                      *int_in_urb;
```

```
+        int                             int_in_running;
+
+        /* 8 byte buffer for the control msg */
+        char                            *ctrl_buffer;
+        struct urb                      *ctrl_urb;
+
+        /* Setup packet information */
+        struct usb_ctrlrequest          *ctrl_dr;
+        int                             correction_required;
+
+        /* Last issued command */
+        unsigned char                   command;
+};
+
+/* Table of devices that work with this driver */
+static struct usb_device_id launcher_table[] =
+{
+        { USB_DEVICE(LAUNCHER_VENDOR_ID, LAUNCHER_PRODUCT_ID) },
+        {} /* Terminating entry */
+};
+MODULE_DEVICE_TABLE (usb, launcher_table);
+
+static struct usb_driver launcher_driver =
+{
+        .name = "launcher_driver",
+        .id_table = launcher_table,
+};
+
+static void launcher_ctrl_callback(struct urb *urb)
+{
+        struct usb_ml *dev = urb->context;
+        pr_debug("launcher_ctrl_callback\n");
+        dev->correction_required = 0;
+}
+
+static void launcher_abort_transfers(struct usb_ml *dev)
+{
+        if (! dev) {
+                pr_err("dev is NULL");
+                return;
+        }
+
+        if (! dev->udev) {
+                pr_err("udev is NULL");
+                return;
+        }
+
+        if (dev->udev->state == USB_STATE_NOTATTACHED) {
+                pr_err("udev not attached");
+                return;
+        }
```

```
+
+        /* Shutdown transfer */
+        if (dev->int_in_running) {
+                dev->int_in_running = 0;
+                mb();
+                if (dev->int_in_urb) {
+                        usb_kill_urb(dev->int_in_urb);
+                }
+        }
+
+        if (dev->ctrl_urb) {
+                usb_kill_urb(dev->ctrl_urb);
+        }
+}
+
+static void launcher_int_in_callback(struct urb *urb)
+{
+        struct usb_ml *dev = urb->context;
+        int retval;
+        int i;
+
+        pr_debug("launcher_int_in_callback\n");
+
+        pr_debug("actual_length: 0x%X - data was: ", urb->actual_length);
+        for (i = 0; i < urb->actual_length; ++i) {
+                pr_debug("0x%X ", (const unsigned char)(urb->
+                                        transfer_buffer) + i);
+        }
+
+        if (urb->status) {
+                if (urb->status == -ENOENT) {
+                        pr_debug("received -NOENT\n");
+                        return;
+                } else if (urb->status == -ECONNRESET) {
+                        pr_debug("received  -ECONNRESET\n");
+                        return;
+                } else if (urb->status == -ESHUTDOWN) {
+                        pr_debug("received -ESHUTDOWN\n");
+                        return;
+                } else {
+                        pr_err("non-zero urb status (%d)", urb->status);
+                        goto resubmit; /* Maybe we can recover. */
+                }
+        }
+
+        if (urb->actual_length > 0) {
+                spin_lock(&dev->cmd_spinlock);
+
+                if (dev->int_in_buffer[0] & LAUNCHER_MAX_UP
+                    && dev->command & LAUNCHER_UP) {
+                        dev->command &= ~LAUNCHER_UP;
```

```
+                       dev->correction_required = 1;
+               } else if (dev->int_in_buffer[0] & LAUNCHER_MAX_DOWN &&
+                          dev->command & LAUNCHER_DOWN) {
+                       dev->command &= ~LAUNCHER_DOWN;
+                       dev->correction_required = 1;
+               }
+
+               if (dev->int_in_buffer[1] & LAUNCHER_MAX_LEFT
+                   && dev->command & LAUNCHER_LEFT) {
+                       dev->command &= ~LAUNCHER_LEFT;
+                       dev->correction_required = 1;
+               } else if (dev->int_in_buffer[1] & LAUNCHER_MAX_RIGHT &&
+                          dev->command & LAUNCHER_RIGHT) {
+                       dev->command &= ~LAUNCHER_RIGHT;
+                       dev->correction_required = 1;
+               }
+
+
+               if (dev->correction_required) {
+                       dev->ctrl_buffer[0] = dev->command;
+                       spin_unlock(&dev->cmd_spinlock);
+                       retval = usb_submit_urb(dev->ctrl_urb, GFP_ATOMIC);
+                       if (retval) {
+                               pr_err(
+                               "submitting correction control URB failed (%d)"
+                               , retval);
+                       }
+               } else {
+                       spin_unlock(&dev->cmd_spinlock);
+               }
+       }
+
+resubmit:
+       /* Resubmit if we're still running. */
+       if (dev->int_in_running && dev->udev) {
+               retval = usb_submit_urb(dev->int_in_urb, GFP_ATOMIC);
+               if (retval) {
+                       pr_err("resubmitting urb failed (%d)", retval);
+                       dev->int_in_running = 0;
+               }
+       }
+}
+
+static inline void launcher_delete(struct usb_ml *dev)
+{
+       launcher_abort_transfers(dev);
+
+       /* Free data structures. */
+       if (dev->int_in_urb) {
+               usb_free_urb(dev->int_in_urb);
+       }
```

```
+        if (dev->ctrl_urb) {
+                usb_free_urb(dev->ctrl_urb);
+        }
+
+        kfree(dev->int_in_buffer);
+        kfree(dev->ctrl_buffer);
+        kfree(dev->ctrl_dr);
+        kfree(dev);
+}
+
+static int launcher_open(struct inode *inodep, struct file *filp)
+{
+        struct usb_ml *dev = NULL;
+        struct usb_interface *interface;
+        int subminor;
+        int retval = 0;
+
+        pr_debug("launcher_open\n");
+        subminor = iminor(inodep);
+
+        mutex_lock(&disconnect_mutex);
+
+        interface = usb_find_interface(&launcher_driver, subminor);
+        if (!interface) {
+                pr_err("can't find device for minor %d", subminor);
+                retval = -ENODEV;
+                goto exit;
+        }
+
+        dev = usb_get_intfdata(interface);
+        if (!dev) {
+                retval = -ENODEV;
+                goto exit;
+        }
+
+        /* lock this device */
+        if (down_interruptible(&dev->sem)) {
+                pr_err("sem down failed");
+                retval = -ERESTARTSYS;
+                goto exit;
+        }
+
+        /* Increment our usage count for the device. */
+        ++dev->open_count;
+        if (dev->open_count > 1) {
+                pr_info("open_count = %d", dev->open_count);
+        }
+
+        /* Initialize interrupt URB. */
+        usb_fill_int_urb(dev->int_in_urb, dev->udev,
+                         usb_rcvintpipe(dev->udev,
```

```
+                           dev->int_in_endpoint->bEndpointAddress),
+                        dev->int_in_buffer,
+                        le16_to_cpu(dev->int_in_endpoint->wMaxPacketSize),
+                        launcher_int_in_callback,
+                        dev,
+                        dev->int_in_endpoint->bInterval);
+
+       dev->int_in_running = 1;
+       mb();
+
+       retval = usb_submit_urb(dev->int_in_urb, GFP_KERNEL);
+       if (retval) {
+               pr_err("submitting int urb failed (%d)", retval);
+               dev->int_in_running = 0;
+               --dev->open_count;
+               goto unlock_exit;
+       }
+
+       /* Save our object in the file's private structure. */
+       filp->private_data = dev;
+
+unlock_exit:
+       up(&dev->sem);
+
+exit:
+       mutex_unlock(&disconnect_mutex);
+       return retval;
+}
+
+static int launcher_close(struct inode *inodep, struct file *filp)
+{      struct usb_ml *dev = NULL;
+       int retval = 0;
+
+       pr_debug("launcher_close\n");
+       dev = filp->private_data;
+
+       if (! dev) {
+               pr_err("dev is NULL");
+               retval =  -ENODEV;
+               goto exit;
+       }
+
+       /* Lock our device */
+       if (down_interruptible(&dev->sem)) {
+               retval = -ERESTARTSYS;
+               goto exit;
+       }
+
+       if (dev->open_count <= 0) {
+               pr_err("device not opened");
+               retval = -ENODEV;
```

```
+                goto unlock_exit;
+        }
+
+        if (! dev->udev) {
+                pr_warn("device unplugged before the file was released");
+
+                /* Unlock here as LAUNCHER_delete frees dev. */
+                up (&dev->sem);
+                launcher_delete(dev);
+                goto exit;
+        }
+
+        if (dev->open_count > 1) {
+                pr_info("open_count = %d", dev->open_count);
+        }
+
+        launcher_abort_transfers(dev);
+        --dev->open_count;
+
+unlock_exit:
+        up(&dev->sem);
+
+exit:
+        return retval;
+}
+
+static ssize_t launcher_read(struct file *f, char __user *buf, size_t cnt,
+                             loff_t *off)
+{
+        int retval = -EFAULT;
+        pr_debug("launcher_read\n");
+        return retval;
+}
+
+static ssize_t launcher_write(struct file *filp, const char __user *user_buf,
+                              size_t count, loff_t *off)
+{
+        int retval = -EFAULT;
+        struct usb_ml *dev;
+        unsigned char buf[8];
+        unsigned char cmd = LAUNCHER_STOP;
+
+        pr_debug("launcher_write\n");
+        dev = filp->private_data;
+
+        /* Lock this object. */
+        if (down_interruptible(&dev->sem)) {
+                retval = -ERESTARTSYS;
+                goto exit;
+        }
+
```

```
+	/* Verify that the device wasn't unplugged. */
+	if (! dev->udev) {
+		retval = -ENODEV;
+		pr_err("No device or device unplugged (%d)", retval);
+		goto unlock_exit;
+	}
+
+	/* Verify that we actually have some data to write. */
+	if (count == 0) {
+		goto unlock_exit;
+	}
+
+	/* We only accept one-byte writes. */
+	if (count != 1) {
+		count = 1;
+	}
+
+	if (copy_from_user(&cmd, user_buf, count)) {
+		retval = -EFAULT;
+		goto unlock_exit;
+	}
+
+	pr_info("Received command 0x%x\n", cmd);
+
+	/* TODO: Check the range of the commands allowed - otherwise we're
+	 *	 trusting the user not to be silly
+	 */
+
+	memset(&buf, 0, sizeof(buf));
+	buf[0] = LAUNCHER_CTRL_COMMAND_PREFIX;
+	buf[1] = cmd;
+
+	/* The interrupt-in-endpoint handler also modifies dev->command. */
+	spin_lock(&dev->cmd_spinlock);
+	dev->command = cmd;
+	spin_unlock(&dev->cmd_spinlock);
+
+	pr_debug("Sending usb_control_message()\n");
+	retval = usb_control_msg(dev->udev,
+				 usb_sndctrlpipe(dev->udev, 0),
+				 LAUNCHER_CTRL_REQUEST,
+				 LAUNCHER_CTRL_REQUEST_TYPE,
+				 LAUNCHER_CTRL_VALUE,
+				 LAUNCHER_CTRL_INDEX,
+				 &buf,
+				 sizeof(buf),
+				 HZ*5);
+
+
+	if (retval < 0) {
+		pr_err("usb_control_msg failed (%d)", retval);
```

```
+                goto unlock_exit;
+        }
+
+        /* We've only written one byte hopefully! */
+        retval = count;
+
+unlock_exit:
+        up(&dev->sem);
+
+exit:
+        return retval;
+}
+
+static struct file_operations fops =
+{
+        .open = launcher_open,
+        .release = launcher_close,
+        .read = launcher_read,
+        .write = launcher_write,
+};
+
+static int launcher_probe(struct usb_interface *interface,
+                          const struct usb_device_id *id)
+{
+        struct usb_device *udev = interface_to_usbdev(interface);
+        struct usb_ml *dev = NULL;
+        struct usb_host_interface *iface_desc;
+        struct usb_endpoint_descriptor *endpoint;
+        int i, int_end_size;
+        int retval = -ENODEV;
+
+        pr_debug("launcher_probe\n");
+
+        /* Set up our class */
+        class.name = LAUNCHER_NODE"%d";
+        class.fops = &fops;
+
+        if ((retval = usb_register_dev(interface, &class)) < 0) {
+                /* Something stopped us from registering this driver */
+                pr_err("Not able to get a minor for this device.");
+                goto exit;
+        } else {
+                pr_info("Minor receieved - %d\n", interface->minor);
+        }
+
+        if (!udev) {
+                /* Something has gone bad */
+                pr_err("udev is NULL");
+                goto exit;
+        }
+
```

```
+	dev = kzalloc(sizeof(struct usb_ml), GFP_KERNEL);
+	if (!dev) {
+		pr_err("cannot allocate memory for struct usb_ml");
+		retval = -ENOMEM;
+		goto exit;
+	}
+
+	dev->command = LAUNCHER_STOP;
+
+	sema_init(&dev->sem, 1);
+	spin_lock_init(&dev->cmd_spinlock);
+
+	dev->udev = udev;
+	dev->interface = interface;
+	iface_desc = interface->cur_altsetting;
+
+	/* Set up interrupt endpoint information. */
+	for (i = 0; i < iface_desc->desc.bNumEndpoints; ++i) {
+		endpoint = &iface_desc->endpoint[i].desc;
+
+		if (((endpoint->bEndpointAddress & USB_ENDPOINT_DIR_MASK)
+		    == USB_DIR_IN)
+				&& ((endpoint->bmAttributes &
+				    USB_ENDPOINT_XFERTYPE_MASK) ==
+					USB_ENDPOINT_XFER_INT))
+			dev->int_in_endpoint = endpoint;
+
+	}
+
+	if (!dev->int_in_endpoint) {
+		pr_err("could not find interrupt in endpoint");
+		goto error;
+	}
+
+	int_end_size = le16_to_cpu(dev->int_in_endpoint->wMaxPacketSize);
+
+	dev->int_in_buffer = kmalloc(int_end_size, GFP_KERNEL);
+	if (!dev->int_in_buffer) {
+		pr_err("could not allocate int_in_buffer");
+		retval = -ENOMEM;
+		goto error;
+	}
+
+	dev->int_in_urb = usb_alloc_urb(0, GFP_KERNEL);
+	if (!dev->int_in_urb) {
+		pr_err("could not allocate int_in_urb");
+		retval = -ENOMEM;
+		goto error;
+	}
+
+	/* Set up the control URB. */
```

```
+        dev->ctrl_urb = usb_alloc_urb(0, GFP_KERNEL);
+        if (!dev->ctrl_urb) {
+                pr_err("could not allocate ctrl_urb");
+                retval = -ENOMEM;
+                goto error;
+        }
+
+        dev->ctrl_buffer = kzalloc(LAUNCHER_CTRL_BUFFER_SIZE, GFP_KERNEL);
+        if (!dev->ctrl_buffer) {
+                pr_err("could not allocate ctrl_buffer");
+                retval = -ENOMEM;
+                goto error;
+        }
+
+        dev->ctrl_dr = kmalloc(sizeof(struct usb_ctrlrequest), GFP_KERNEL);
+        if (!dev->ctrl_dr) {
+                pr_err("could not allocate usb_ctrlrequest");
+                retval = -ENOMEM;
+                goto error;
+        }
+        dev->ctrl_dr->bRequestType = LAUNCHER_CTRL_REQUEST_TYPE;
+        dev->ctrl_dr->bRequest = LAUNCHER_CTRL_REQUEST;
+        dev->ctrl_dr->wValue = cpu_to_le16(LAUNCHER_CTRL_VALUE);
+        dev->ctrl_dr->wIndex = cpu_to_le16(LAUNCHER_CTRL_INDEX);
+        dev->ctrl_dr->wLength = cpu_to_le16(LAUNCHER_CTRL_BUFFER_SIZE);
+
+        usb_fill_control_urb(dev->ctrl_urb, dev->udev,
+                        usb_sndctrlpipe(dev->udev, 0),
+                        (unsigned char *)dev->ctrl_dr,
+                        dev->ctrl_buffer,
+                        LAUNCHER_CTRL_BUFFER_SIZE,
+                        launcher_ctrl_callback,
+                        dev);
+
+        /* Retrieve a serial. */
+        if (!usb_string(udev, udev->descriptor.iSerialNumber,
+                        dev->serial_number,
+                                sizeof(dev->serial_number))) {
+                pr_err("could not retrieve serial number");
+                goto error;
+        }
+
+        /* Save our data pointer in this interface device. */
+        usb_set_intfdata(interface, dev);
+
+        dev->minor = interface->minor;
+
+exit:
+        return retval;
+
+error:
```

```
+        launcher_delete(dev);
+        return retval;
+}
+
+static void launcher_disconnect(struct usb_interface *interface)
+{
+        struct usb_ml *dev;
+        int minor;
+
+        pr_debug("launcher_disconnect\n");
+        mutex_lock(&disconnect_mutex);        /* Not interruptible */
+
+        dev = usb_get_intfdata(interface);
+        usb_set_intfdata(interface, NULL);
+
+        down(&dev->sem); /* Not interruptible */
+
+        minor = dev->minor;
+
+        /* Give back our minor. */
+        usb_deregister_dev(interface, &class);
+
+        /* If the device is not opened, then we clean up right now. */
+        if (! dev->open_count) {
+                up(&dev->sem);
+                launcher_delete(dev);
+        } else {
+                dev->udev = NULL;
+                up(&dev->sem);
+        }
+
+        mutex_unlock(&disconnect_mutex);
+
+}
+
+static int __init launcher_init(void)
+{
+        int result;
+
+        pr_debug("launcher_init\n");
+
+        /* Wire up our probe/disconnect */
+        launcher_driver.probe = launcher_probe;
+        launcher_driver.disconnect = launcher_disconnect;
+
+        /* Register this driver with the USB subsystem */
+        if ((result = usb_register(&launcher_driver))) {
+                pr_err("usb_register() failed. Error number %d", result);
+        }
+        return result;
+}
```

```
+
+static void __exit launcher_exit(void)
+{
+        pr_debug("launcher_exit\n");
+        /* Deregister this driver with the USB subsystem */
+        usb_deregister(&launcher_driver);
+}
+
+module_init(launcher_init);
+module_exit(launcher_exit);
+
+MODULE_LICENSE("GPL");
+MODULE_AUTHOR("Nick Glynn <Nick.Glynn@feabhas.com>");
+MODULE_DESCRIPTION("USB Launcher Driver");
--- /home/geoff/school/linux/drivers/usb/Kconfig        2012-10-08 07:32:26.228555000 -0700
+++ kernel/drivers/usb/Kconfig        2013-06-05 22:23:32.000000000 -0700
@@ -51,6 +51,12 @@
         # more:
         default PCI

+config USB_MISSILE
+        tristate "Dream cheeky usb missile launcher (Thunder model)"
+        default m
+        ---help---
+        Generic driver for dream cheeky usb missile launcher
+
 # some non-PCI hcds implement EHCI
 config USB_ARCH_HAS_EHCI
         boolean
--- /home/geoff/school/linux/drivers/usb/Makefile        2012-10-08 07:32:26.228555000 -0700
+++ kernel/drivers/usb/Makefile        2013-06-05 22:00:24.000000000 -0700
@@ -51,3 +51,5 @@
 obj-$(CONFIG_USB_RENESAS_USBHS)        += renesas_usbhs/
 obj-$(CONFIG_USB_OTG_UTILS)        += otg/
 obj-$(CONFIG_USB_GADGET)        += gadget/
+
+ojb-$(CONFIG_USB_MISSILE)         += ml.o
```

```
#define _POSIX_C_SOURCE 2 //shutsup compiler
#define _BSD_SOURCE //shuts up compiler

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define LAUNCHER_NODE "/dev/launcher0"
#define LAUNCHER_FIRE 0x10
#define LAUNCHER_STOP 0x20
#define LAUNCHER_UP 0x02
#define LAUNCHER_DOWN 0x01
```

```
#define LAUNCHER_LEFT 0x04
#define LAUNCHER_RIGHT 0x08
#define LAUNCHER_UP_LEFT (LAUNCHER_UP | LAUNCHER_LEFT)
#define LAUNCHER_DOWN_LEFT (LAUNCHER_DOWN | LAUNCHER_LEFT)
#define LAUNCHER_UP_RIGHT (LAUNCHER_UP | LAUNCHER_RIGHT)
#define LAUNCHER_DOWN_RIGHT (LAUNCHER_DOWN | LAUNCHER_RIGHT)

static void launcher_cmd(int fd, int cmd)
{
        int retval = 0;

        retval = write(fd, &cmd, 1);
        if (retval < 0) {
                fprintf(stderr, "Could not send command to " LAUNCHER_NODE
                        " (error %d)\n", retval);
        } else if (LAUNCHER_FIRE == cmd) {
                usleep(5000000);
        }
}

static void launcher_usage(char *name)
{
        fprintf(stderr, "Usage: %s [-mfslrudh] [-t <msecs>]\n"
                        "\t-m\tmissile launcher [" LAUNCHER_NODE "]\n"
                        "\t-f\tfire\n"
                        "\t-s\tstop\n"
                        "\t-l\tturn left\n"
                        "\t-r\tturn right\n"
                        "\t-u\tturn up\n"
                        "\t-d\tturn down\n"
                        "\t-t\tspecify duration to wait before sending STOP in milliseconds\n"
                        "\t-h\tdisplay this help\n\n"
                        "Notes:\n"
                        "\tIt is possible to combine the directions of the two axis, e.g.\n"
                        "\t'-lu' send_cmds the missile launcher up and left at the same time.\n"
                        "" , name);
        exit(1);
}


int main(int argc, char **argv)
{
        char c;
        int fd;
        int cmd = LAUNCHER_STOP;
        char *dev = LAUNCHER_NODE;
        unsigned int duration = 500;

        if (argc < 2) {
                launcher_usage(argv[0]);
        }
```

```
        while ((c = getopt(argc, argv, "mlrudfsht:")) != -1) {
                switch (c) {
                case 'm':
                        dev = optarg;
                        break;
                case 'l':
                        cmd = LAUNCHER_LEFT;
                        break;
                case 'r':
                        cmd = LAUNCHER_RIGHT;
                        break;
                case 'u':
                        cmd = LAUNCHER_UP;
                        break;
                case 'd':
                        cmd = LAUNCHER_DOWN;
                        break;
                case 'f':
                        cmd = LAUNCHER_FIRE;
                        break;
                case 's':
                        cmd = LAUNCHER_STOP;
                        break;
                case 't':
                        duration = strtol(optarg, NULL, 10);
                        fprintf(stdout, "Duration set to %d\n", duration);
                        break;
                default:
                        launcher_usage(argv[0]);
                }
        }

        fd = open(dev, O_RDWR);
        if (fd == -1) {
                perror("Couldn't open file: %m");
                exit(1);
        }
        launcher_cmd(fd, cmd);
        usleep(duration * 1000);
        launcher_cmd(fd, LAUNCHER_STOP);
        close(fd);
        return EXIT_SUCCESS;
}
```