CS411: Individual Writeup - Group 13
Project 5: USB Rocket Launcher

Trevor Bramwell

December 2, 2012

Introduction

The main point of this assignment was to learn how to write a USB device driver for the Linux kernel, understand how the USB protocol works at a low level, and write a userspace program that interacts with a driver.

Our group achieved this primarily by using a preexisting driver and controller program we found online written by Nick Glynn. Nick had previously completed a project similar to this one.

Approach

I approached this assignment with high hopes that completing it would be very simple. Before I started I was already aware of the *Linux Device Drivers, 3rd Edition* (LDD3) book by Greg Kroah-Hartman, et al. which outlines how to write device drivers for the Linux kernel. Upon actually reading through the USB device driver chapter of LDD3 I realized that it would not be as great of an asset as I originally though.

LDD3 is currently completely outdated. Though the general overview on USB was somewhat informative, the rest of the chapter read primarily as a reference guide to USB structs in the Linux kernel. Thus, I needed to look else where – le Internet – for more information.

Design

Since Matt ended up finding a pre-written driver and controller program for the USB Dream Cheeky Rocket Launcher, not much else was required for me to do besides test our resources.

Testing

Testing for this project consisted of building the driver against the current kernel. Loading the module into the Linux kernel, and then running the userspace program.

This was done by using/modifying the example **Makefile** from LDD3 for compiling modules outside the Linux tree. It was similar to this:

```
# If KERNELRELEASE is defined, we've been invoked from the
# kernel build system and can use its language.
ifneq ($(KERNELRELEASE),)

    obj-m := hello.o

# Otherwise we were called directly from the command
# line; invoke the kernel build system.
else

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

```
default:

    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif
```

once that was written to a file on Squidly 13, making the module was as simple as running *make*.

After the module was made, it needed to be loaded into the kernel. This was accomplished by running:

```
sudo insmod launcher_driver.ko
```

I then ran the userspace program compiled from *launcher_control.c* to move and fired the USB rocket launcher. This worked well, but didn't allow easy control, so I edited it to continually grab character from the user and send those 'commands' directly to the launcher.

launcher_control.c

```
#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)

#define SEC 1000

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 = 100;

        fd = open(dev, O_RDWR);
        if (fd == -1) {
                perror("Couldn't open file: %m");
                exit(1);
        }

        /*while ((c = getopt(argc, argv, "mlrudfsht:")) != -1) {*/
        system("/bin/stty raw");
        while ((c = getchar()) != 'q') {
                switch (c) {
                        /*
                case 'm':
                        dev = optarg;
                        break;
                        */
                case 'a':
                        cmd = LAUNCHER_LEFT;
                        break;
                case 'd':
                        cmd = LAUNCHER_RIGHT;
                        break;
                case 'w':
                        cmd = LAUNCHER_UP;
                        break;
                case 's':
                        cmd = LAUNCHER_DOWN;
                        break;
                case ' ':
                        cmd = LAUNCHER_FIRE;
                        break;
                case 'e':
                        cmd = LAUNCHER_STOP;
                        break;
                        /*
                case 't':
                        duration = strtol(optarg, NULL, 10);
                        fprintf(stdout, "Duration set to %d\n", duration);
                        break;
                        */
                }

                launcher_cmd(fd, cmd);
                usleep(duration * SEC);
                launcher_cmd(fd, LAUNCHER_STOP);
        }

        close(fd);
        system("/bin/stty cooked");
        return EXIT_SUCCESS;
}
```

Learning

Since getting the USB Rocket Launcher up and running was no very challenging, because our group didn't actually write any code for the driver, most of our time was used reading about how USB drivers work and how to implement them.

As with the Shortest Seek Time First elevator algorithm, there was already a predefined interface to work with in the kernel. This interface required that the USB device driver implement the functions:

- open
- release – remapped as close
- read
- write

These were implemented in the driver that we got from Nick Glynn. This is what each of them did.

Open

Open the device for access. Called when an application wants access to the device through udev or the */dev* directory. Similar to opening a file.

Release

Release the device from being accessed. Similar to closing a file.

Read

Read from the device. This does nothing for the USB Rocket Launcher as it does not send back any data that is useful to a user.

Write

Write to the device. This is the meat of the USB Rocket Launcher driver as it handled sending all the commands to the device and setting flags for responses.

Conclusion

Overall this was a fun assignment as each of us learned how to work with devices that don't have pre-existing drivers, and we got to shoot each other.