Ben Williams

Lab 7

CprE 308

The purpose of this lab was to learn about basic linux kernel module development as well as the usage of user space linux. We started the lab by downloading, unzipping and installing the linux 3.18-8 kernel. We then verified the version of linux and looked around the more important directories.

**In your report list five architectures that the Linux kernel supports**

Apple Ax

Broadcom VideoCore

Samsung Exynos

iPAQ

Qualcomm Snapdragon

**In your report list three filesystem types Linux supports**

minix

ext

ext2

**Include the following outputs in your lab report**

root@HOSTNAME:/Lab7# cd hello-world/

root@HOSTNAME:/Lab7/hello-world# ls

hello\_world.c hello\_world.mod.c Makefile

hello\_world.c~ hello\_world.mod.o modules.order

hello\_world.ko hello\_world.o Module.symvers

root@HOSTNAME:/Lab7/hello-world# lsmod

Module Size Used by

root@HOSTNAME:/Lab7/hello-world# modinfo hello\_world.ko

filename: /Lab7/hello-world/hello\_world.ko

description: A simple hello world driver

author: Jeramie Vens vens@iastate.edu

license: GPL

depends:

vermagic: 3.18.8NETID mod\_unload

root@HOSTNAME:/Lab7/hello-world# insmod hello\_world.ko

I just got back from a cruise!

root@HOSTNAME:/Lab7/hello-world# lsmod

Module Size Used by

hello\_world 811 0

root@HOSTNAME:/Lab7/hello-world# rmmod hello\_world.ko

This class is challenging!

root@HOSTNAME:/Lab7/hello-world# lsmod

Module Size Used by

root@HOSTNAME:/Lab7/hello-world#

**The prototype of this function is static ssize\_t read(struct file \*file, char \*buffer, size\_t length, loff\_t \*offset). You should notice the similarity of this prototype to the system call read which has prototype ssize\_t read(int fd, void \*buf, size\_t count);. In your lab report comment on why the prototypes are not identical and what the extra paramaters are for (use your intuition to help answer this question).**

These prototypes are not identical because this function needs the additional offset information in the fourth argument. There may be an offset between the driver's expectations of where data is and where it actually is. Since the regular read() command is internal, it doesn't need to worry about offset since all information is organized according to linux convention.

**In your report list six operations that files can support.**

Read

Write

Save

**Before running the module please read through the code and make sure you understand what it should do. In your report give your expected output when the file is opened, written to, read from, and closed.**

The module basically creates a character device using register\_chrdev() with name “cpre308”. If this is successful, it prints out diagnostics about the device such as the major number (typically identifying the driver associated with the device.) It shows where the device's communications file is located under dev.

Then, when the device is opened, the user is shown how many times hello world has been printed. The program trys to get the module, whichs simply returns true if the module is not being presently removed.

While the device is being read, the program simply loops and prints the next byte (character) from the port into user space using put\_user(). When there are no more characters left (length == 0) or the message pointer points to null, the loop terminates and the function returns the number of bytes read.

Device being written to is not supported.

**In your lab report include the major number of the module. In your labreport mention the output of each of these:**

**root@HOSTNAME:/Lab7/hello-file# insmod hello\_file.ko**

hello\_file: module license 'unspecified' taints kernel.

Disabling lock debugging due to kernel taint

I was assigned major number 254. To talk to

the driver, create a dev file with

'mknod /dev/cpre308 c 254 0'.

Try various minor numbers. Try to cat and echo to

the device file.

**root@HOSTNAME:/Lab7/hello-file# mknod /dev/cpre308-0 c 254 0**

mknod: ‘/dev/cpre308-0’: File exists

**root@HOSTNAME:/Lab7/hello-file# cat /dev/cpre308-0**

I already told you 0 times Hello World!

**root@HOSTNAME:/Lab7/hello-file# cat /dev/cpre308-0**

I already told you 1 times Hello World!

**root@HOSTNAME:/Lab7/hello-file# echo "hello" > /dev/cpre308**

**root@HOSTNAME:/Lab7/hello-file# mknod /dev/cpre308-1 c 254 1**

**root@HOSTNAME:/Lab7/hello-file# cat /dev/cpre308-0**

I already told you 2 times Hello World!

**root@HOSTNAME:/Lab7/hello-file# cat /dev/cpre308-1**

I already told you 3 times Hello World!

**root@HOSTNAME:/Lab7/hello-file# rmmod hello\_file.ko**

**root@HOSTNAME:/Lab7/hello-file# cat /dev/cpre308-0**

cat: /dev/cpre308-0: No such device or address

**Printer Module**

I then proceeded to create the printer driver as specified in part 5.3. I used get\_user() to get information from the file being passed in character by character, and checked each byte if it was a new line. Once I saw the first new line, I knew that I had gotten to the end of the first file, and therefore my buffer had the complete name of the print job. I then used kprint to inform the user that I had gotten the job name and printed the job name.