ECE 373 – Assignment #2
Character Driver

Check all of these materials into your Github Classroom repo before **11pm, Pacifc time, Thursday, 25-April-2024**

Objectives:

It's time to start playing with the device driver interface in the kernel. This assignment will use information from the previous lectures and combine that with this week's character device lecture. Roll all that into one, and we can start sending data into and retrieving data from the kernel. A sample C file is uploaded to the Modules section in CANVAS to get you started.

1. Setting up the char device (30 pts):

The first part of this assignment deals with getting the character device framework built. Create a kernel module that creates and registers a single character device (`struct cdev`). Make sure this device dynamically allocates a `dev_t` major number; statically allocating a device will result in loss of points.

Make sure to also document how to create the device file in `/dev` to operate on. This will need to be in the list of things to turn in with the assignment. This part of the assignment requires you to implement the creation on /dev file in two different ways:

(a) Using the umount command as we did in the class.

(b) Use the driver code to create a /sys entry using class_create function and then create a /dev entry using device_create function. You can use LXR, LDD book or internet to search for the correct syntax to use class_create and device_create functions. Comment your code with a brief description of what these functions do. Make sure to correctly handle the errors these function calls might throw.

When creating the device, make sure if has user access permissions (chmod command can be used).

2. Hooking up file operations (20 pts)

By now you should have a shiny kernel module that loads and does nothing useful. Let's make it useful:

Add an integer variable called "`syscall_val`" to your char device struct. Initialize this value to 42 to begin with.

Write a `read()` callback function that will read the variable "`syscall_val`" and return it to userspace (remember `copy_to_user`?). Make sure to have a `printk()` to print that you're in the `read()` system call.

Write a `write()` callback function that will write a new value to "`syscall_val`". You do not need to do data verification at this point, but you're more than welcome to (remember `copy_from_user`?). Make sure to have a `printk()` to print that you're in the `write()` system call.

3. Testing, testing... (25 pts):

Now your driver is hooked up and ready to accept inputs. Write a tiny C program in userspace that opens the character device you created (it's ok to hard-code your `/dev` filename here), reads the current value of the "`syscall_val`" variable, and displays it. Then have your C program write a new value to change "`syscall_val`". After writing the value, read the value back out and display it.

Make sure your module unloads successfully, and properly cleans up ALL resources you allocated in the driver (`chardev`, `dev_t`, etc.).

4. Changes (25 pts):

Suppose you want to let your users specify their favorite initial value – instead of '42', maybe they want to start with '97'. Modify your program to use a module parameter to set the starting value. Reload the module, setting the module parameter value on the command line. Use your test program to read and print and change the initial value. Lastly, using the '`cat`' shell command, read the current value of the module parameter from the sysfs entry (/sys/module/…).

Debugging Hint:

Be sure to check return values in your userspace test program to catch errors and print any resulting error values. See the man pages for `open()`, `read()`, `write()`, `errno.h`, and possibly even `perror()` for more information.

What to turn in:

1. Source code to your kernel module, plus your kernel module Makefile.
2. Your userspace program. This does not require a Makefile, just the source code.
3. A typescript of loading the driver and running your userspace program both before and after adding the module parameter.
4. A typescript of `/proc/devices` showing your module loaded