ELEC 424/553

Mobile & Embedded Systems

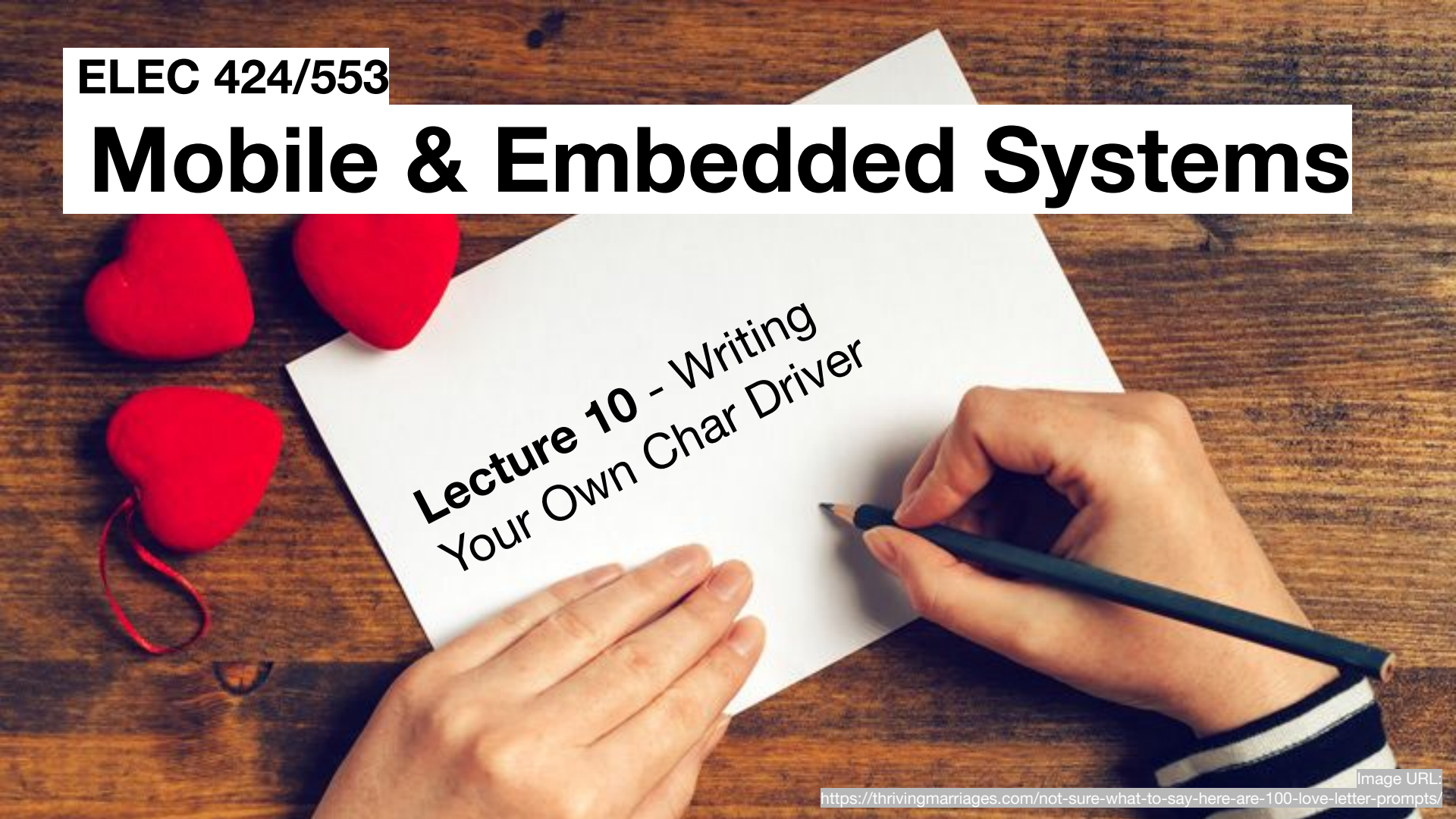
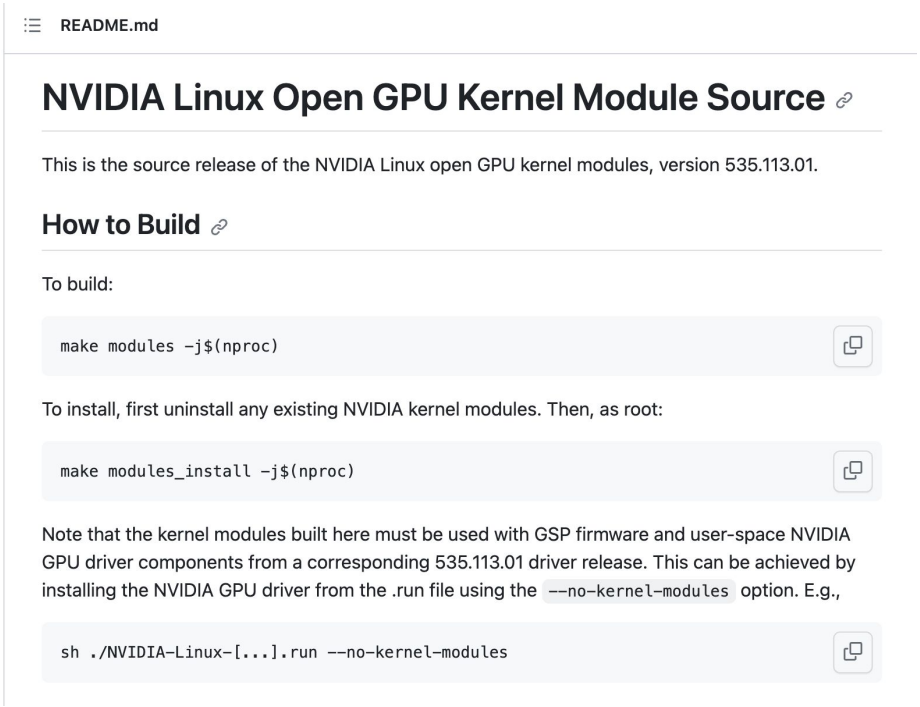


Image URL:
https://thrivingmarriages.com/not-sure-what-to-say-here-are-100-love-letter-prompts/

Why Character Devices?

From YouTube channel DubStepZz. URL: https://www.youtube.com/watch?v=YisYNBBd3Jw

NVIDIA Open Source Driver - https://github.com/NVIDIA/open-gpu-kernel-modules



README.md

NVIDIA Linux Open GPU Kernel Module Source

This is the source release of the NVIDIA Linux open GPU kernel modules, version 535.113.01.

How to Build

To build:

```
make modules -j$(nproc)
```

To install, first uninstall any existing NVIDIA kernel modules. Then, as root:

```
make modules_install -j$(nproc)
```

Note that the kernel modules built here must be used with GSP firmware and user-space NVIDIA GPU driver components from a corresponding 535.113.01 driver release. This can be achieved by installing the NVIDIA GPU driver from the .run file using the `--no-kernel-modules` option. E.g.,

```
sh ./NVIDIA-Linux-[...].run --no-kernel-modules
```

Housekeeping

- **Assignment 1 - Kernel Hacking**
  - Add your own system call and use it
  - Due next Wednesday October 4th at 11:59pm
  - NOTE: If you prefer to submit a video demo, please upload it to the Box folder that I shared with your Rice email.
    - **It is your responsibility to let me know 48 hours before the deadline if you do not have access to the folder, otherwise it is not my responsibility if you have to submit late.**
    - Video files submitted must be named as the following (replacing words as appropriate): ELEC424_Assignment1_Firstname_Lastname_netID
- **Again: Bring Raspberry Pi to every future class**
  - Let me know after class today if this is an issue

Don’t Do This At Home!

- Recommend not using your primary laptop and operating system
  - However, you are unlikely to encounter issues
  - Just be sure to save anything else running on your computer and be ready for a random crash; Don’t hold me liable for this!
- Reportedly does not work on WSL, and will not work on CLEAR (you need sudo)
- Use Raspberry Pi, VirtualBox, or some other way of protecting your main system and files

The Code You'll See Today Is a Combination of:

- Derek Molloy's (Dr. Derek Molloy, School of Electronic Engineering, Dublin City University, Ireland) excellent work here:
  http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/
  http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/
- Corbet, Rubini, & Kroah-Hartman, Linux Device Drivers, 3rd Ed. URL: https://lwn.net/Kernel/LDD3/
- My own **craziness**

init & exit

init

- insmod runs this

exit

- rmmod runs this

Log Levels (KERN_INFO) For printk()

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abraham Lincoln");
MODULE_DESCRIPTION("Greatest module in the world!");
MODULE_VERSION("0.000001");

static int __init hello_init(void){
    printk(KERN_INFO "Oh hi mark\n");
    return 0;
}

static void __exit hello_exit(void){
    printk(KERN_INFO "sad, but still love you\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Adding a Module Parameter

- **`insmod`** can include multiple parameter values
- Example from LDD book:

```
insmod hello.ko howmany=10 whom="Mom"
```

- In the module, we use a macro: `module_param`
  - See **`moduleparam.h`** here
- Three inputs to **`module_param`**:
  - Variable name
  - Variable type
  - Permissions mask

/ include / linux / moduleparam.h

```
101  /**
102   * module_param - typesafe helper for a module/cmdline parameter
103   * @name: the variable to alter, and exposed parameter name.
104   * @type: the type of the parameter
105   * @perm: visibility in sysfs.
106   *
107   * @name becomes the module parameter, or (prefixed by KBUILD_MODNAME and a
108   * ".") the kernel commandline parameter.  Note that - is changed to _, so
109   * the user can use "foo-bar=1" even for variable "foo_bar".
110   *
111   * @perm is 0 if the variable is not to appear in sysfs, or 0444
112   * for world-readable, 0644 for root-writable, etc.  Note that if it
113   * is writable, you may need to use kernel_param_lock() around
114   * accesses (esp. charp, which can be kfreed when it changes).
115   *
116   * The @type is simply pasted to refer to a param_ops_##type and a
117   * param_check_##type: for convenience many standard types are provided but
118   * you can create your own by defining those variables.
119   *
120   * Standard types are:
121   *      byte, hexint, short, ushort, int, uint, long, ulong
122   *      charp: a character pointer
123   *      bool: a bool, values 0/1, y/n, Y/N.
124   *      invbool: the above, only sense-reversed (N = true).
125   */
126  #define module_param(name, type, perm)                          \
127          module_param_named(name, name, type, perm)
```

Linux Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/latest/source/include/linux/moduleparam.h

Permissions Mask

- Our third field for **`module_param`**
- Check out **`stat.h`** - link
  - Also check out other stat.h file here (not pictured)
- Header gives definitions for permissions macros
- **`S_IRUGO`** - Anyone can read (can't modify)
- **`S_IRUGO | S_IWUSR`** - Anyone can read; Modifiable by root
- Can navigate to **`/sys/module`** to view parameter

/ include / linux / stat.h

```
1   /* SPDX-License-Identifier: GPL-2.0 */
2   #ifndef _LINUX_STAT_H
3   #define _LINUX_STAT_H
4
5
6   #include <asm/stat.h>
7   #include <uapi/linux/stat.h>
8
9   #define S_IRWXUGO       (S_IRWXU|S_IRWXG|S_IRWXO)
10  #define S_IALLUGO       (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
11  #define S_IRUGO         (S_IRUSR|S_IRGRP|S_IROTH)
12  #define S_IWUGO         (S_IWUSR|S_IWGRP|S_IWOTH)
13  #define S_IXUGO         (S_IXUSR|S_IXGRP|S_IXOTH)
```

Linux Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/latest/source/include/linux/stat.h

Let's Add a Parameter (multiplier) Via module_param()

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abraham Lincoln");
MODULE_DESCRIPTION("Greatest module in the world!");
MODULE_VERSION("0.000001");

static int multiplier = 10;
module_param(multiplier, int, S_IRUGO);

static int __init hello_init(void){
    printk(KERN_INFO "Oh hi mark\n");
    return 0;
}

static void __exit hello_exit(void){
    printk(KERN_INFO "sad, but still love you\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

We Need to Print Our Beautiful Parameter (Because We Love it So Much)

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abraham Lincoln");
MODULE_DESCRIPTION("Greatest module in the world!");
MODULE_VERSION("0.000001");

static int multiplier = 10;
module_param(multiplier, int, S_IRUGO);

static int __init hello_init(void){
    printk(KERN_INFO "Oh hi mark - I love Lisa %dX more than you do\n", multiplier);
    return 0;
}

static void __exit hello_exit(void){
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multiplier);
}

module_init(hello_init);
module_exit(hello_exit);
```

Makefile (Unchanged)

```
obj-m+=hello.o

all:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

Try it out!

```
sudo insmod hello.ko multiplier=3

ls /sys/module/

ls /sys/module/hello/

ls /sys/module/hello/parameters/

cat /sys/module/hello/parameters/multiplier

sudo rmmod hello.ko
```

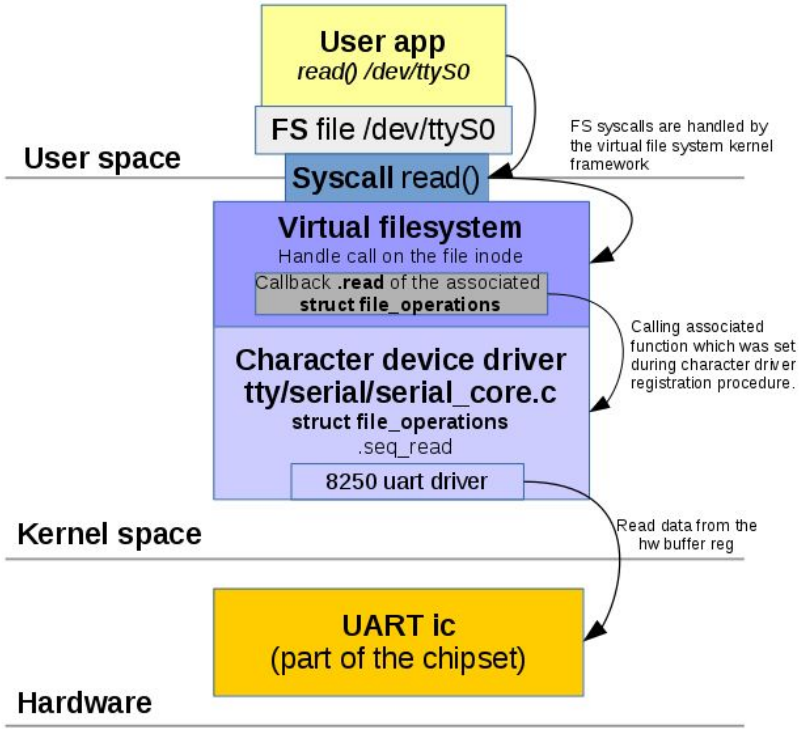
Uploaded Example Files

- See Canvas/Examples/Simple\ Module [the \ is intentional]
  - **`hello.c`**
  - **`Makefile`**

Making a Character (Char) Device Driver

Character (char) Devices (*cdevs*)

- Interfaced through **stream of bytes**
- System calls commonly used include:
  - `open`
  - `close`
  - `read`
  - `write`
- Similar to **files**
  - But generally limited to sequential access
- Appear in `/dev`
- Majority of basic devices can be accessed via char drivers



Corbet, Rubini, & Kroah-Hartman, *Linux Device Drivers*, 3rd Ed.
Figure: Oleg Kutkov, "Simple Linux character device driver". URL: https://olegkutkov.me/2018/03/14/simple-linux-character-device-driver/

Demonstration [Notes for Joe]

- Joe goes to Pi
- Inserts module for exercise
- ls /dev
  - Note ttyS0
- ls -al /dev [note c/b/d]
- ls -al ~/hello_module [note - for regular files]
- Demo expected functionality of module with insmod, ./test, rmmod, and dmesg

System Calls & Drivers

- User applications can only interact with hardware through system calls to the kernel
- **Drivers implement system calls for devices**
- Our goal today is to write basic code to implement the **`open`** system call in our driver

file_operations

/ include / linux / fs.h All symb Search Identifier

```
2022  struct file_operations {
2023          struct module *owner;
2024          loff_t (*llseek) (struct file *, loff_t, int);
2025          ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
2026          ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
2027          ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
2028          ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
2029          int (*iopoll)(struct kiocb *kiocb, bool spin);
2030          int (*iterate) (struct file *, struct dir_context *);
2031          int (*iterate_shared) (struct file *, struct dir_context *);
2032          __poll_t (*poll) (struct file *
2033          long (*unlocked_ioctl) (struct
2034          long (*compat_ioctl) (struct fi
2035          int (*mmap) (struct file *, struct vm_area_struct *);
2036          unsigned long mmap_supported_flags;
2037          int (*open) (struct inode *, struct file *);
2038          int (*flush) (struct file *, fl_owner_t id);
2039          int (*release) (struct inode *, struct file *);
2040          int (
2041          int (
2042          int (
```

Function pointer

Exercise 9 Goal: Implement `open` in our driver

Linux Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h

Major Number, Class

- `ls /sys/class/mes`
  - `ls /sys/class/mes/meschar`

User Interaction

- We need a user application to call **`open`** for a fake character device we will play with - you will do this in exercise 9
  - Make a C file called testmeschar.c
  - Include the header files stdio.h, stdlib.h, unistd.h, and fcntl.h
  - Inside the main function, print (using printf) a message to the console saying that we are running the program
  - Make a call to the function open()
    - open(), which is part of the standard C library, takes two input arguments
    - The first input is a string that specifies the file path to our fake device we are making, which is at /dev/**** [where **** is the device name you chose]
    - The second argument is O_RDWR
      - Capital O, not zero; Used to read/write files
    - The output will be an integer; No need to do anything with it
  - return 0

Open Requires Having The Device At /dev

- Writing a user-space C application that accesses our fake device via our driver through the C open() function (which uses the open system call)
- Device name will determine what appears at /dev
  - E.g., /dev/COOL_GPU
- Choose device name (fill in X)
  - #define DEVICE_NAME “X”
- Choose class name (fill in X)
  - #define CLASS_NAME “X”
- Then can access using open(“/dev/COOL_GPU”, O_RDWR)
  - O_RDWR is a macro

Other Functions In Your Driver

init

- register_chrdev(0, DEVICE_NAME, &fops)
- class_create(THIS_MODULE, CLASS_NAME)
- device_create(mescharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);

exit

- device_destroy(mescharClass, MKDEV(majorNumber,0));
- class_unregister(mescharClass);
- class_destroy(mescharClass);
- unregister_chrdev(majorNumber, DEVICE_NAME);

Exercise 9: Make new `Makefile`

```
obj-m+=hello.o

all:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
    $(CC) INSERT_TEST_C_FILE_NAME_WITH_EXTENSION -o test
clean:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
    rm test
```