ELEC 424/553

Mobile & Embedded Systems

Lecture 11 - Writing Your Own Char Driver (2)

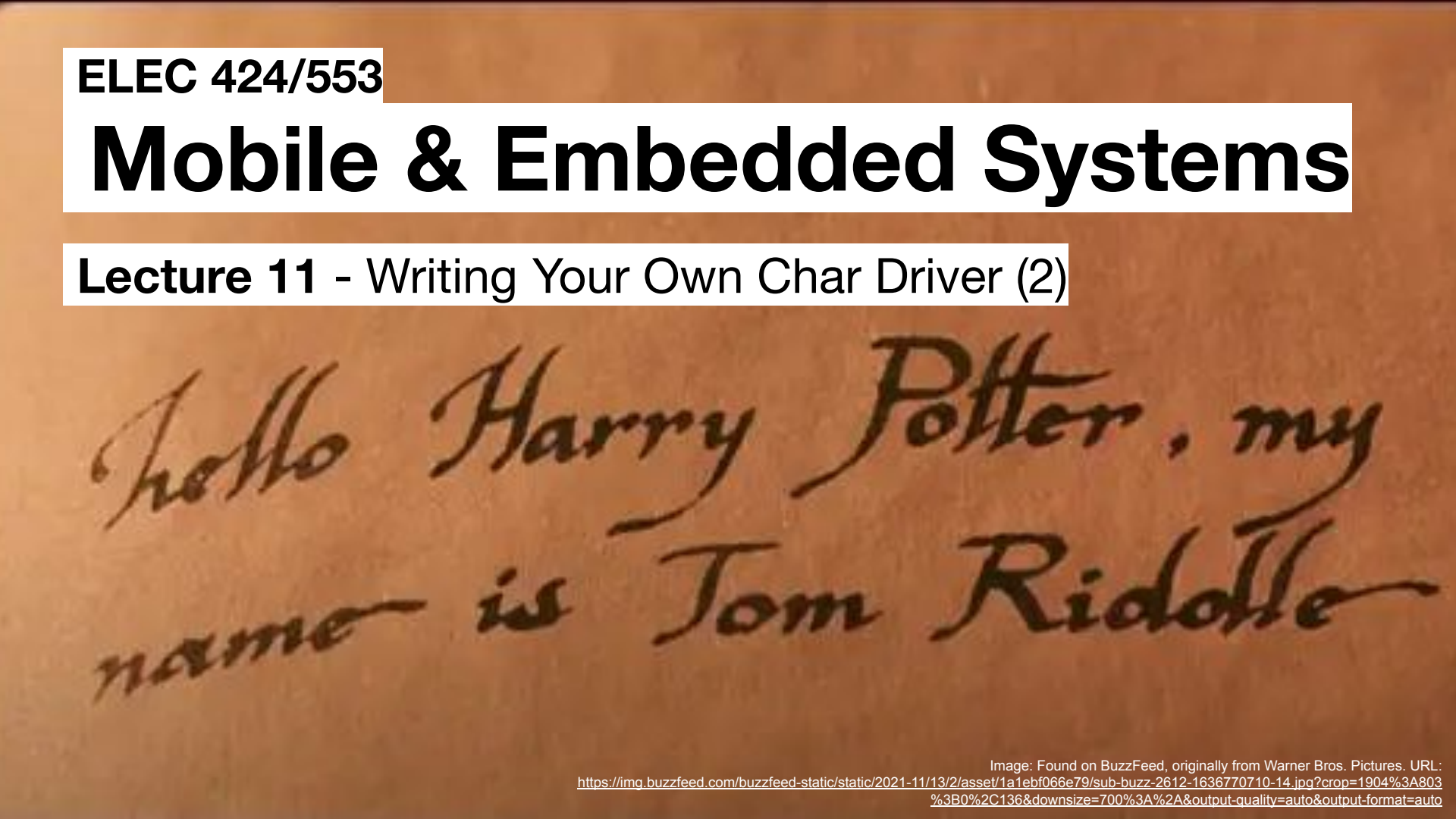


Image: Found on BuzzFeed, originally from Warner Bros. Pictures. URL: https://img.buzzfeed.com/buzzfeed-static/static/2021-11/13/2/asset/1a1ebf066e79/sub-buzz-2612-1636770710-14.jpg?crop=1904%3A803%3B0%2C136&downsize=700%3A%2A&output-quality=auto&output-format=auto

Thinking about a master's degree? Get an MECE

Rice MECE Webinars - Hosted by Joe Young, Mike Orchard, and Nyetta Meaux

Wed Oct 11th, 8:30-9:30am (CST)

- Join Zoom Meeting
- ID: 91774958501
- Passcode: 303185

Tue Nov 14th, 7:30-8:30pm (CST)

- Join Zoom Meeting
- ID: 96422456703
- Passcode: 247293

As a Rice student, you can also have the application fee waived and potentially receive a tuition waiver for TA service or potentially be awarded a scholarship.

Email Joe at jy46@rice.edu with any questions

Functions In Your Driver

init

- **register_chrdev(0, DEVICE_NAME, &fops)**
  - *Provides major number for driver*
    - Multiple devices can use same driver (with same major number) but will each have different minor numbers
  - <u>1st arg</u>: Desired major number - 0 means we will take whatever we are given (“dynamic”)
  - <u>2nd arg</u>: Can be arbitrary - not used for /dev/device_name [instead has to do with device owner]
  - <u>3rd arg</u>: file_operations struct to be linked between system call and driver

Functions In Your Driver

init

- **class_create(THIS_MODULE, CLASS_NAME)**
  - *Create class of device*
    - Can group together devices in /sys/class
  - 1st arg: Class owner module
  - 2nd arg: - Class name :)

Functions In Your Driver

init

- **device_create(mescharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME)**

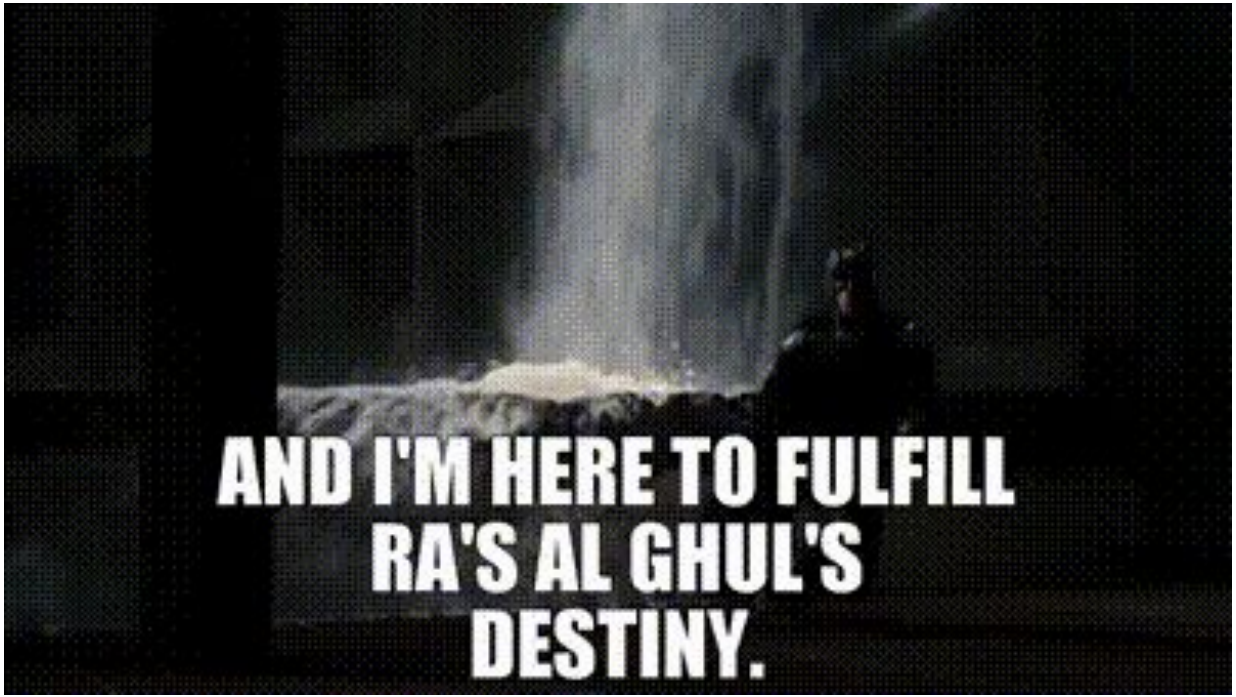
The following is quoted from the Linux source code from Bootlin Elixir - URL: https://elixir.bootlin.com/linux/latest/source/drivers/base/core.c#L4325:

```
“* device_create - creates a device and registers it with sysfs
 * @class: pointer to the struct class that this device should be registered to
 * @parent: pointer to the parent struct device of this new device, if any
 * @devt: the dev_t for the char device to be added
 * @drvdata: the data to be added to the device for callbacks
 * @fmt: string for the device's name”
```

MKDEV(majorNumber, 0) - 1st arg is major number, 2nd arg is minor number

Finishing What We Started

exit



From *The Dark Knight Rises*. Warner Bros. 2012. Nolan, C. Found at URL: https://getyarn.io/yarn-clip/aabac5b0-870e-484d-a3f9-8c0823c5409f

Functions In Your Driver

exit

- **device_destroy(mescharClass, MKDEV(majorNumber,0))**
  - Undoes device_create - destroys device
- *class_unregister(mescharClass)*
  - Actually, this is called by class_destroy
- **class_destroy(mescharClass)**
  - Removes class
- **unregister_chrdev(majorNumber, DEVICE_NAME)**
  - Unassign major number to driver

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

- [Primarily] 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/
- [Primarily] My own **craziness**
- [Somewhat] The Linux Kernel Module Programming Guide: https://tldp.org/LDP/lkmpg/2.4/html/c577.htm
- [More referential] Corbet, Rubini, & Kroah-Hartman, Linux Device Drivers, 3rd Ed. URL: https://lwn.net/Kernel/LDD3/

Note: Copying Things To Raspberry Pi

From a terminal window not logged into the RPi, use SCP to copy files from your computer to the RPi

- `scp hello.c Makefile test.c pi@raspberrypi.local:~/char_rw_driver`

If you want to copy from the RPi to your computer:

- `scp pi@raspberrypi.local:~/char_rw_driver/Makefile .`

Let’s walk through the code - here is where we started

```
#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);
```

Add On To Hi Mark Module: Header Files

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.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);
```

Add DEVICE_NAME and CLASS_NAME

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

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);
}
```

Add More Variables

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;

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;
}
```

Add Prototype Function For Function To Use For open

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;

static int device_open(struct inode *, struct file *);

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;
```

Link open System Call With Driver device_open

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;

static int device_open(struct inode *, struct file *);

static struct file_operations fops =
{
      .open = device_open
};

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

static int multiplier = 10;
```

Overhaul hello_init

```
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){
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    mescharClass = class_create(THIS_MODULE, CLASS_NAME);
    mescharDevice = device_create(mescharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
    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);
```

Overhaul hello_exit

```
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){
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    mescharClass = class_create(THIS_MODULE, CLASS_NAME);
    mescharDevice = device_create(mescharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
    printk(KERN_INFO "Oh hi mark - I love Lisa %dX more than you do\n", multiplier);
    return 0;
}

static void __exit hello_exit(void){
    device_destroy(mescharClass, MKDEV(majorNumber,0));
    class_unregister(mescharClass);
    class_destroy(mescharClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multiplier);
}

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

Define device_open

```
}

static void __exit hello_exit(void){
    device_destroy(mescharClass, MKDEV(majorNumber,0));
    class_unregister(mescharClass);
    class_destroy(mescharClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multiplier);
}

static int device_open(struct inode *inodep, struct file *filep){
    printk(KERN_INFO "You’re tearing me apart, Lisa!\n");
    return 0;
}

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

User Interaction

- We need a user application to call **`open`** for this device
- This will also require an addition to the **`Makefile`**

Create New File: `testmeschar.c`

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

int main(){
      int fd;
      printf("Warm it up.exe\n");
      fd = open("/dev/meschar", O_RDWR);      // Capital o, not zero
      return 0;
}
```

From GIFER. URL: https://i.gifer.com/YgY.gif

Modify Makefile

```
obj-m+=hello.o

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

Try It Out!

- `make`
- `sudo insmod hello.ko`
- `ls /dev`
- `ls /dev/mes*` (* is a wildcard, different from a Yellowcard)
- `ls /sys/class/mes`
  - `ls /sys/class/mes/meschar`
- `cat /sys/module/hello/parameters/multiplier`
- (other terminal window) `tail -f /var/log/kern.log`
  - Or: `dmesg`
- `sudo ./test` (run many times) [MAKE SURE TO RUN WITH sudo!]
- `sudo rmmod hello.ko`

Let's Make device_open Actually Do Something - Add Open Counter

```
1  #include <linux/module.h>
2  #include <linux/init.h>
3  #include <linux/kernel.h>
4  #include <linux/fs.h>
5  #include <linux/device.h>
6  #include <linux/uaccess.h>
7  #define DEVICE_NAME "meschar"
8  #define CLASS_NAME "mes"
9
10 static int majorNumber;
11 static struct class* mescharClass = NULL;
12 static struct device* mescharDevice = NULL;
13 static int timesCalled = 0;
14
15 static int device_open(struct inode *, struct file *);
16
17 static struct file_operations fops =
18 {
19   .open = device_open
20 };
21
```

Increment Open Counter in device_open

```
static void __exit hello_exit(void){
    device_destroy(mescharClass, MKDEV(majorNumber,0));
    class_unregister(mescharClass);
    class_destroy(mescharClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multiplier);
}

static int device_open(struct inode *inodep, struct file *filep){
    timesCalled++;
    printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
    return 0;
}

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

Try It Out

- **`sudo insmod hello.ko`**
- (other terminal window) **`tail -f /var/log/kern.log`**
- **`sudo ./test`** (run many times)
- **`sudo rmmod hello.ko`**

file_operations

open() c function in user space → system call handler in kernel space → fops struct → open() function in driver

/ include / linux / fs.h Identifier

```
2022  struct file_operations {
2023          struct module *owner;
2024          loff_t (*llseek) (struct file *, loff_
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 fi
2032          __poll_t (*poll) (struct file *,
2033          long (*unlocked_ioctl) (struct f                    ng);
2034          long (*compat_ioctl) (struct fil                    );
2035          int (*mmap) (struct file *, stru
2036          unsigned long map_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 (*fsync) (struct file *, loff_t, loff_t, int datasync);
2041          int (*fasync) (int, struct file *, int);
2042          int (*lock) (struct file *, int, struct file_lock *);
```

Now let's implement **read** and **write**

Implemented **open** last time

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

Goal: Interact With User String Using `read` and `write`

- `write`
  - Allow user to enter string
  - Save the string
- `read`
  - Print out the saved string

Considerations: The user is not always our friend.

Also, what could go wrong in a concurrent situation?

Picture from *Harry Potter*. Warner Bros. Found at URL: https://screenrant.com/harry-potter-best-moments-harry-ron-friendship/

Defining The Problem: A Lack of Synchronization

‘The Room - Flower Shop Scene (Blu Ray) [1080p]’. From *The Room* (2003). URL: https://www.youtube.com/watch?v=KIkoXhgtI58

Also Remember: We Don't Trust User Applications, Ever

/ include / linux / uaccess.h

All syr Sear

```
188  static __always_inline unsigned long __must_check
189  copy_from_user(void *to, const void __user *from, unsigned long n)
190  {
191          if (likely(check_copy_size(to, n, false)))
192                  n = _copy_from_user(to, from, n);
193          return n;
194  }
195
196  static __always_inline unsigned long __must_check
197  copy_to_user(void __user *to, const void *from, unsigned long n)
198  {
199          if (likely(check_copy_size(from, n, true)))
200                  n = _copy_to_user(to, from, n);
201          return n;
202  }
```

Linux Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/latest/source/include/linux/uaccess.h#L189

Add On To fops struct

```
17 static struct file_operations fops =
18 {
19    .open = device_open,
20    .read = device_read,
21    .write = device_write,
22 };
23
24 MODULE_LICENSE("GPL");
25 MODULE_AUTHOR("Abraham Lincoln");
26 MODULE_DESCRIPTION("Greatest module in the world!");
27 MODULE_VERSION("0.000001");
28
29 static int multiplier = 10;
30 module_param(multiplier, int, S_IRUGO);
31
32 static int __init hello_init(void){
33     majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
34     mescharClass = class_create(THIS_MODULE, CLASS_NAME);
35     mescharDevice = device_create(mescharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
36     printk(KERN_INFO "Oh hi mark - I love Lisa %dX more than you do\n", multiplier);
37     return 0;
38 }
39
40 static void __exit hello_exit(void){
```

We can put a comma here, no problem

This is good practice since the code may be modified later and someone will forget to put a comma

file_operations

write() or read() c function in user space → system call handler in kernel space → fops struct → write() or read() function in driver

/ 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 *, struct poll_table_struct *);
2033         long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
2034         long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
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 (*fsync) (struct file *, loff_t, loff_t, int datasync);
2041         int (*fasync) (int, struct file *, int);
2042         int (*lock) (struct file *, int, struct file_lock *);
```

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

Add Skeletons For New Functions

```
47
48 static int device_open(struct inode *inodep, struct file *filep){
49     timesCalled++;
50     printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
51     return 0;
52 }
53
54 static ssize_t device_read(){
55
56 }
57
58 static ssize_t device_write(){
59
60 }
61
62
63 module_init(hello_init);
64 module_exit(hello_exit);
```

Add Input Argument Types For New Functions

```
47
48 static int device_open(struct inode *inodep, struct file *filep){
49     timesCalled++;
50     printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
51     return 0;
52 }
53
54 static ssize_t device_read(struct file *, char __user *, size_t, loff_t *){
55
56 }
57
58 static ssize_t device_write(struct file *, const char __user *, size_t, loff_t *){
59
60 }
61
62
63 module_init(hello_init);
64 module_exit(hello_exit);
```

Give Names To Input Arguments For New Functions

```
47
48 static int device_open(struct inode *inodep, struct file *filep){
49     timesCalled++;
50     printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
51     return 0;
52 }
53
54 static ssize_t device_read(struct file *filep, char __user *buf, size_t length, loff_t *offset){
55
56 }
57
58 static ssize_t device_write(struct file *filep, const char __user *buf, size_t length, loff_t *offset){
59
60 }
61
62
63 module_init(hello_init);
64 module_exit(hello_exit);
```

Let's Make device_write Do Something

```
47
48 static int device_open(struct inode *inodep, struct file *filep){
49     timesCalled++;
50     printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
51     return 0;
52 }
53
54 static ssize_t device_read(struct file *filep, char __user *buf, size_t length, loff_t *offset){
55
56 }
57
58 static ssize_t device_write(struct file *filep, const char __user *buf, size_t length, loff_t *offset){
59    long error_count;
60    printk("Running device_write\n");
61    error_count = copy_from_user(message,buf,length);
62    size_of_message = strlen(message);
63    printk(KERN_INFO "mesChar: Received %d characters from the user\n", size_of_message);
64    printk(KERN_INFO "Message received: %s\n", message);
65    return length;
66 }
67
68
69 module_init(hello_init);
70 module_exit(hello_exit);
```

Add return to device_read

```
1
2 static int device_open(struct inode *inodep, struct file *filep){
3     timesCalled++;
4     printk(KERN_INFO "You're tearing me apart, Lisa! Also I've been opened %d times.\n", timesCalled);
5     return 0;
6 }
7
8 static ssize_t device_read(struct file *filep, char __user *buf, size_t length, loff_t *offset){
9    return 0;
10 }
11
12 static ssize_t device_write(struct file *filep, const char __user *buf, size_t length, loff_t *offset){
13    long error_count;
14    printk("Running device_write\n");
15    error_count = copy_from_user(message,buf,length);
16    size_of_message = strlen(message);
17    printk(KERN_INFO "mesChar: Received %d characters from the user\n", size_of_message);
18    printk(KERN_INFO "Message received: %s\n", message);
19    return length;
20 }
21
22
23 module_init(hello_init);
24 module_exit(hello_exit);
```

Need To Add `message` & `size_of_message`

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;
static int timesCalled = 0;
static char message[256] = {0};
static short size_of_message;

static int device_open(struct inode *, struct file *);

static struct file_operations fops =
{
    .open = device_open,
    .read = device_read,
    .write = device_write,
};
```

Also Need To Add Function Prototypes

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;
static int timesCalled = 0;
static char message[256] = {0};
static short size_of_message;

static int device_open(struct inode *, struct file *);
static ssize_t device_read(struct file *, char __user *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char __user *, size_t, loff_t *);

static struct file_operations fops =
{
    .open = device_open,
    .read = device_read,
    .write = device_write,
```

Now you need to update your testing file to send a string to the driver using the write c function

Hint: Use

```
scanf("%[^\n]%*c", stringToSend);
```

for user input