ELEC 424/553

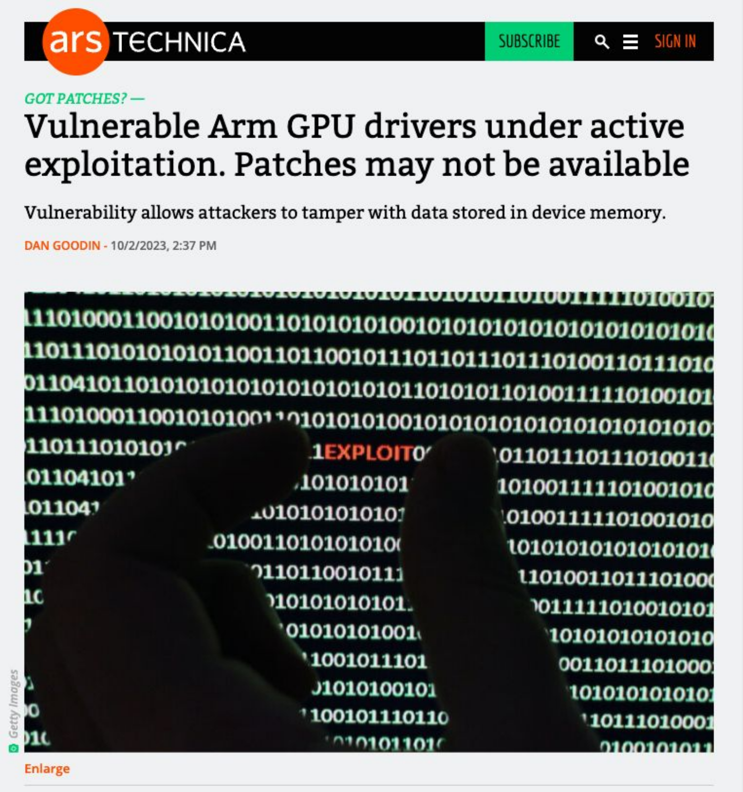
Mobile & Embedded Systems

Lecture 12

Locks & GPIO Intro

Image URL: https://news.unl.edu/newsrooms/today/article/red-green-show-added-to-lied-center-lineup/

In Recent News



GOT PATCHES? —

Vulnerable Arm GPU drivers under active exploitation. Patches may not be available

Vulnerability allows attackers to tamper with data stored in device memory.

DAN GOODIN - 10/2/2023, 2:37 PM

‘Arm warned on Monday of active ongoing attacks targeting a **vulnerability in device drivers** for its Mali line of GPUs, which run on a host of devices, including Google Pixels and other Android handsets, Chromebooks, and **hardware running Linux**.

“A local non-privileged user can make improper GPU memory processing operations to gain access to already freed memory,” Arm officials wrote in an advisory. “This issue is fixed in Bifrost, Valhall and Arm 5th Gen GPU Architecture Kernel Driver r43p0. There is evidence that this vulnerability may be under limited, targeted exploitation. Users are recommended to upgrade if they are impacted by this issue.”

The advisory continued: “A local non-privileged user can make improper GPU processing operations to access a limited amount outside of buffer bounds or to **exploit a software race condition**. If the system’s memory is carefully prepared by the user, then this in turn could give them access to already freed memory.”’

Ars Technica. URL: https://arstechnica.com/security/2023/10/vulnerable-arm-gpu-drivers-under-active-exploitation-patches-may-not-be-available/

`dev` and `sysfs`

- `dev`
  - `/dev`
  - Focused on accessing devices
- `sysfs`
  - Actually a **virtual file system**
  - Files realized on demand
  - In-memory
  - `/sys`
  - Focused on device management
  - Way for user to view & modify **kernel objects**
  - User view of Linux Device Model
- UNIX philosophy: “Everything is a file”

https://www.facebook.com/itsfoss/photos/well-everything-is-actually-a-file-in-unix-/1337947193012711/

/dev and /sys

“The kernel provides a representation of its model in userspace through the sysfs virtual file system. It is usually mounted in the /sys directory and contains the following subdirectories:

- **block** - all block devices available in the system (disks, partitions)
- **bus** - types of bus to which physical devices are connected (pci, ide, usb)
- **class** - drivers classes that are available in the system (net, sound, usb)
- **devices** - the hierarchical structure of devices connected to the system
- **firmware** - information from system firmware (ACPI)
- **fs** - information about mounted file systems
- **kernel** - kernel status information (logged-in users, hotplug)
- **module** - the list of modules currently loaded
- **power** - information related to the power management subsystem”

“Linux Kernel Teaching”. URL: https://linux-kernel-labs.github.io/refs/heads/master/labs/device_model.html

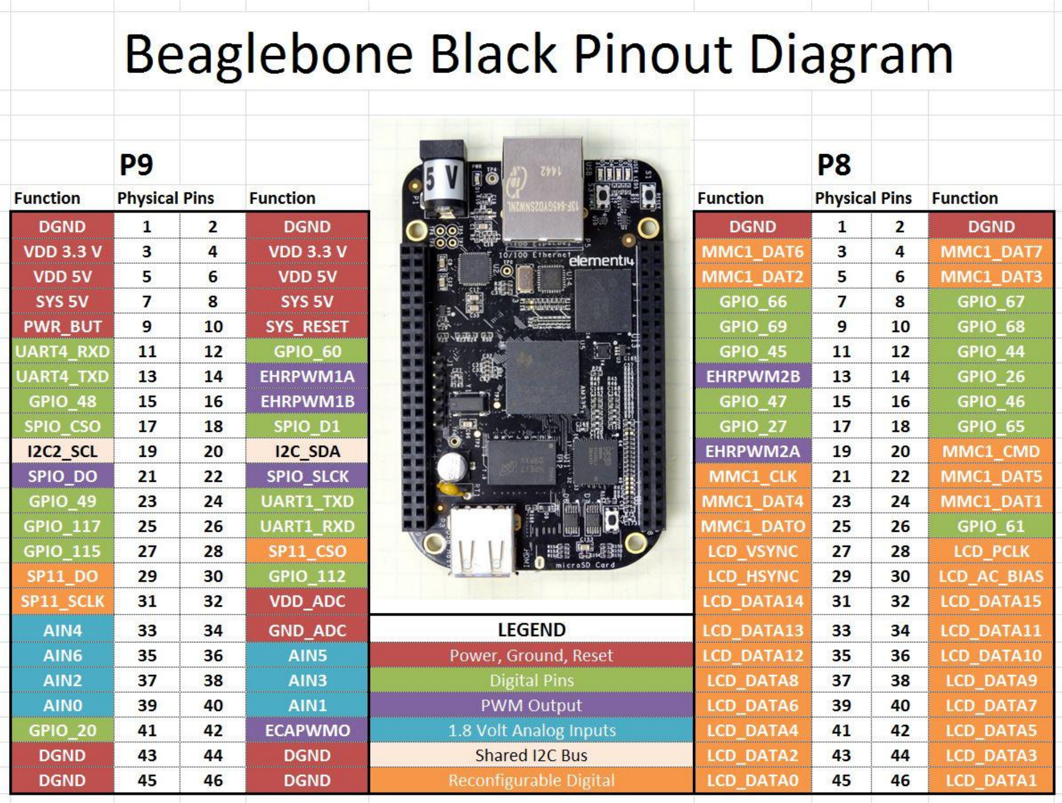
The Linux Device Model

- 2.6 kernel (Dec. 2003) introduced **unified device model**
- Enabled easy view of devices & device hierarchy
- Driver & device association (both ways)
- Cluster devices according to class
  - E.g. “input device”
- But really, why?
  - Power management
  - Need to know what to shut off first
  - USB Mouse -> USB Controller -> PCI Bus

Robert Love, *Linux Kernel Development*, 3rd Edition

Where Are We Headed?

BeagleBone Black GPIO: General-Purpose Input/Output



Beaglebone Black Pinout Diagram

P9

Function	Physical Pins		Function
DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V
VDD 5V	5	6	VDD 5V
SYS 5V	7	8	SYS 5V
PWR_BUT	9	10	SYS_RESET
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPIO_CSO	17	18	SPIO_D1
I2C2_SCL	19	20	I2C_SDA
SPIO_DO	21	22	SPIO_SLCK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SP11_CSO
SP11_DO	29	30	GPIO_112
SP11_SCLK	31	32	VDD_ADC
AIN4	33	34	GND_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPWMO
DGND	43	44	DGND
DGND	45	46	DGND

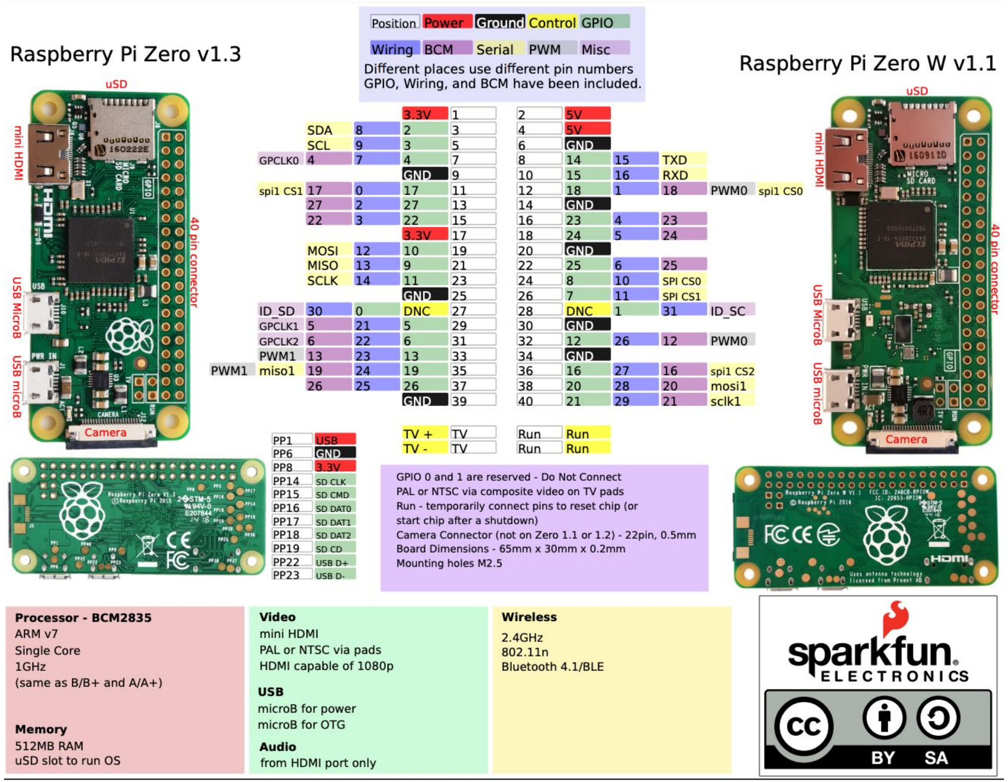
LEGEND
Power, Ground, Reset
Digital Pins
PWM Output
1.8 Volt Analog Inputs
Shared I2C Bus
Reconfigurable Digital

P8

Function	Physical Pins		Function
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DATO	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

“PINOUT FOR BEAGLEBONE BLACK”.
URL: https://toptechboy.com/beaglebone-black-lesson-1-understanding-beaglebone-black-pinout/beaglebone-black-pinout/

Raspberry Pi Zero W GPIO - Want to make drivers using GPIO



Position Power Ground Control GPIO

Wiring BCM Serial PWM Misc

Different places use different pin numbers
GPIO, Wiring, and BCM have been included.

				3.3V	1	2	5V				
		SDA	8	2	3	4	5V				
		SCL	9	3	5	6	GND				
	GPCLK0	4	7	4	7	8	14	15	TXD		
				GND	9	10	15	16	RXD		
	spi1 CS1	17	0	17	11	12	18	1	18	PWM0	spi1 CS0
		27	2	27	13	14	GND				
		22	3	22	15	16	23	4	23		
				3.3V	17	18	24	5	24		
		MOSI	12	10	19	20	GND				
		MISO	13	9	21	22	25	6	25		
		SCLK	14	11	23	24	8	10	SPI CS0		
				GND	25	26	7	11	SPI CS1		
	ID_SD	30	0	DNC	27	28	DNC	1	31	ID_SC	
	GPCLK1	5	21	5	29	30	GND				
	GPCLK2	6	22	6	31	32	12	26	12	PWM0	
	PWM1	13	23	13	33	34	GND				
PWM1	miso1	19	24	19	35	36	16	27	16	spi1 CS2	
		26	25	26	37	38	20	28	20	mosi1	
				GND	39	40	21	29	21	sclk1	
				TV +	TV	Run	Run				
				TV -	TV	Run	Run				

PP1	USB
PP6	GND
PP8	3.3V
PP14	SD CLK
PP15	SD CMD
PP16	SD DAT0
PP17	SD DAT1
PP18	SD DAT2
PP19	SD CD
PP22	USB D+
PP23	USB D-

GPIO 0 and 1 are reserved - Do Not Connect
PAL or NTSC via composite video on TV pads
Run - temporarily connect pins to reset chip (or start chip after a shutdown)
Camera Connector (not on Zero 1.1 or 1.2) - 22pin, 0.5mm
Board Dimensions - 65mm x 30mm x 0.2mm
Mounting holes M2.5

Processor - BCM2835
ARM v7
Single Core
1GHz
(same as B/B+ and A/A+)

Memory
512MB RAM
uSD slot to run OS

Video
mini HDMI
PAL or NTSC via pads
HDMI capable of 1080p

USB
microB for power
microB for OTG

Audio
from HDMI port only

Wireless
2.4GHz
802.11n
Bluetooth 4.1/BLE

From SparkFun. License: CC BY-SA (link). No changes made. URL: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf

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/

Exercise 9 Solution

Original test 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;
}
```

Include <string.h> & write()

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

int main(){
   int fd;
   int ret;
   char stringToSend[256];
   printf("Warm it up.exe\n");
   fd = open("/dev/meschar", O_RDWR);       // Capital o, not zero
   printf("Do you know anything about the Chamber of Secrets?\n");
   scanf("%[^\n]%*c", stringToSend);
   ret = write(fd, stringToSend, strlen(stringToSend));
   return 0;
}
```

Test It Out!

- In one terminal window:
  - `sudo rmmod hello.ko` [in case the module has been previously inserted but not removed]
  - `sudo insmod hello.ko`
- Open another terminal window (viewing the two terminal windows side by side):
  - `tail -f /var/log/kern.log`
- In the first terminal window:
  - `sudo ./test`
- What happens if we run it again with a shorter message?

Let’s Add Capability to device_read

```
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;
}

static ssize_t device_read(struct file *filep, char __user *buf, size_t length, loff_t *offset){
    long error_count;                                 // copy_to_user returns how many bytes failed to copy
    error_count = copy_to_user(buf,message,size_of_message);   // copy_to_user(dest, src, byte length)
    printk("Sent %d characters back\n",size_of_message);
    return 0;
}

static ssize_t device_write(struct file *filep, const char __user *buf, size_t length, loff_t *offset){
    long error_count;
    printk("Running device_write\n");
    error_count = copy_from_user(message,buf,length);
    size_of_message = strlen(message);
    printk(KERN_INFO "mesChar: Received %d characters from the user\n", size_of_message);
    printk(KERN_INFO "Message received: %s\n", message);
    return length;
}

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

testmeschar.c

Add receive & read call to testmeschar.c

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

static char receive[256];

int main(){
   int fd;
   int ret;
   char stringToSend[256];
   printf("Warm it up.exe\n");
   fd = open("/dev/meschar", O_RDWR);      // Capital o, not zero
   printf("Do you know anything about the Chamber of Secrets?\n");
   scanf("%[^\n]%*c", stringToSend);
   ret = write(fd, stringToSend, strlen(stringToSend));
   printf("HP, I'm going to repeat back what you said (if you hit enter)");
   getchar();
   ret = read(fd, receive, 256);
   printf("REPEAT OF MESSAGE: %s\n", receive);
   return 0;
}
```

Test It Out!

- Run **make**
- In first terminal window:
  - `sudo rmmod hello.ko` [in case the module has been previously inserted but not removed]
  - `sudo insmod hello.ko`
- In other terminal window:
  - `tail -f /var/log.kern.log`
- In first terminal window:
  - `sudo ./hello`
- What happens if we run multiple instances of `./hello` at the same time?
  - Processes can be preempted
  - No protection for this in the current code

Potential Issues

- File open incremented by anyone
- String can be overwritten
- Can read string of other processes, either by read or not entering anything for write
- Similarly, parts of string not overwritten will still appear

The Answer: Mutex/Locks

- **Thread synchronization** - “a mechanism which **ensures** that two or more **concurrent processes** or threads **do not simultaneously execute** some particular program segment known as a **critical section**”

  From GeeksforGeeks, “Mutex lock for Linux Thread Synchronization”. URL: https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/

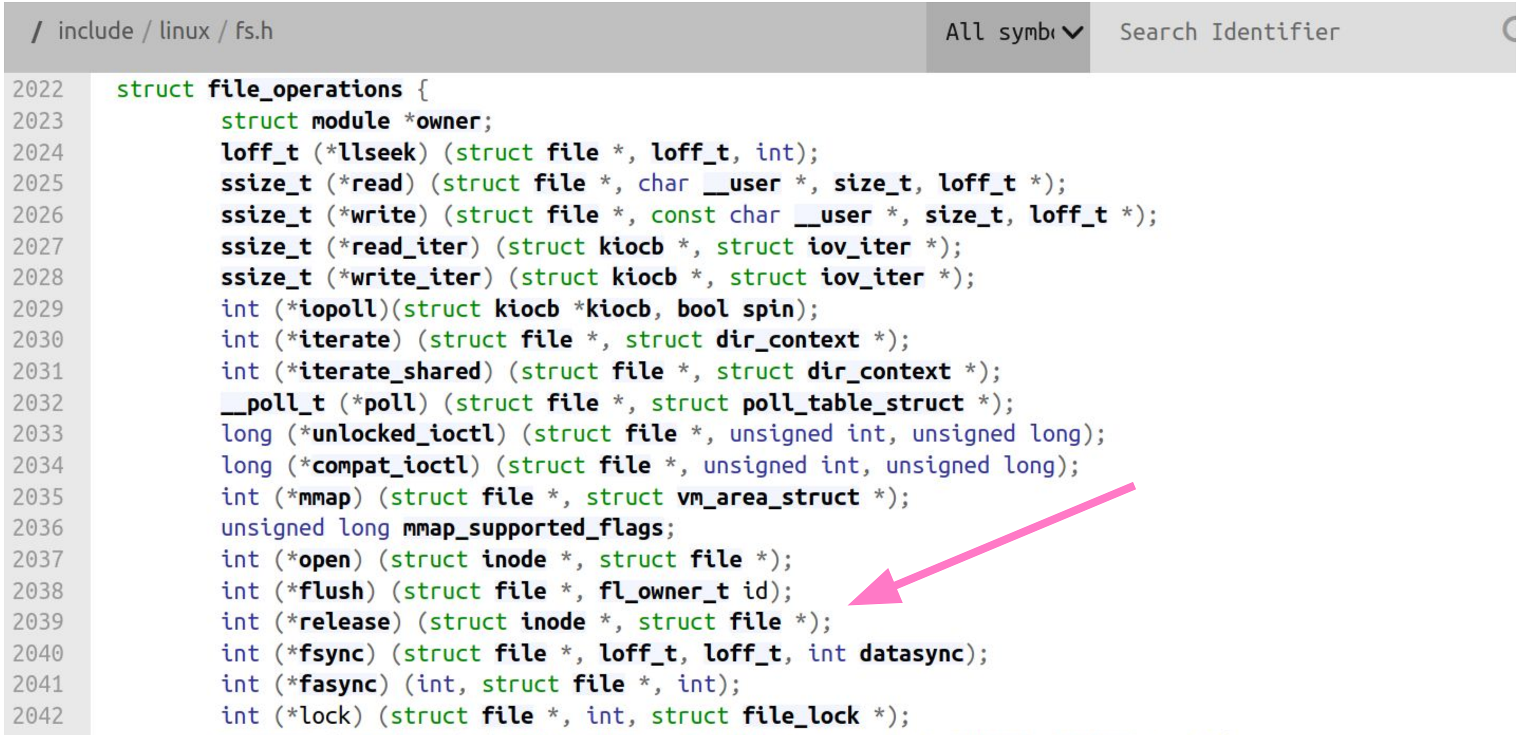
- **Race conditions** at play
  - Non-deterministic what will happen when
  - We saw this!
  - You could have two system call handlers using the open function of the driver
    - Who will win? (Both could see being_used as 0)
- **Mutex** - Lock to guarantee exclusive access to shared resource

Can Try a Basic Locking Approach Using a Static int

- Check locking variable in open, then increment locking variable, and then decrement in new release function
  - Open should return -1 or -EBUSY if driver being used by another application
- Need to add **`release`** function to driver
  - Prototype of release

    -
      ```
      int (*release) (struct inode *, struct file *);
      ```
  - Add release to file_operations
  - (Look at **`file_operations`** struct in **`fs.h`**: https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h)
- Release should alter locking variable

file_operations



```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iopoll)(struct kiocb *kiocb, bool spin);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
        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

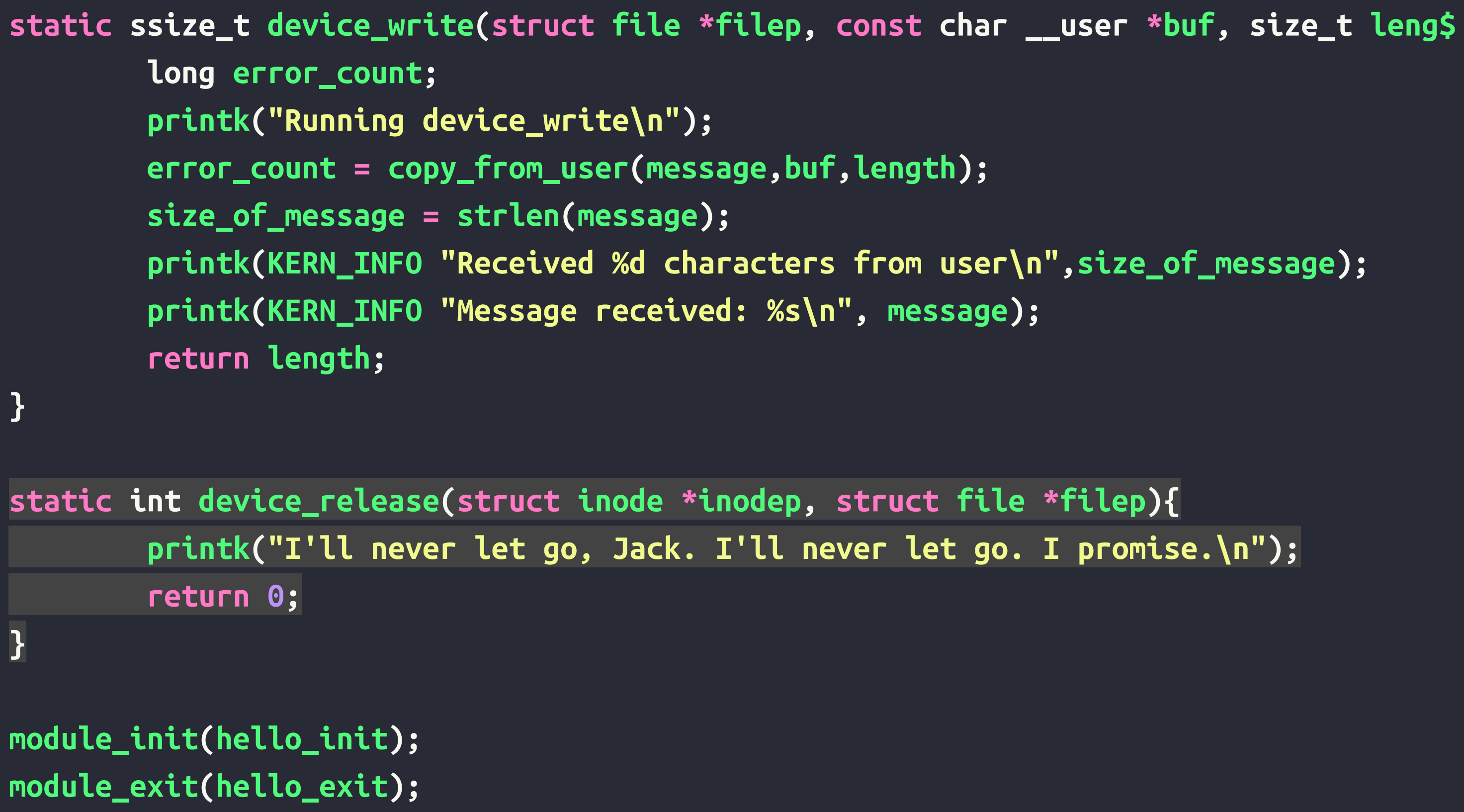
Add Prototype for device_release

```
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 int device_release(struct inode *, struct file *);

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

Add Pointer for device_release

```
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 int device_release(struct inode *, struct file *);

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

Define device_release



```
static ssize_t device_write(struct file *filep, const char __user *buf, size_t leng$
        long error_count;
        printk("Running device_write\n");
        error_count = copy_from_user(message,buf,length);
        size_of_message = strlen(message);
        printk(KERN_INFO "Received %d characters from user\n",size_of_message);
        printk(KERN_INFO "Message received: %s\n", message);
        return length;
}

static int device_release(struct inode *inodep, struct file *filep){
        printk("I'll never let go, Jack. I'll never let go. I promise.\n");
        return 0;
}

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

Now to the Test File!

Add Call to Close (Release) In `testmeschar.c`

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

static char receive[256];

int main(){
        int fd;
        int ret;
        char stringToSend[256];
        printf("Warm it up.exe\n");
        fd = open("/dev/meschar", O_RDWR);       // Capital o, not zero
        printf("Do you know anything about the Chamber of Secrets?\n");
        scanf("%[^\n]%*c", stringToSend);
        ret = write(fd, stringToSend, strlen(stringToSend));
        printf("HP, I'll repeat what you said if you hit enter");
        getchar();
        ret = read(fd, receive, 256);
        printf("REPEAT OF MESSAGE: %s\n", receive);
        close(fd);
        return 0;
}
```

Why are we using close for release?

Need open and close to match [extra detail: http://www.makelinux.net/ldd3/chp-3-sect-5.shtml]

make Again; Then Test It Out

Make Sure Close/Release Message Appears

What happens if you remove the driver, comment out close, recompile, and test again?

Do you still see the close message?

Why?

Add being_used In hello.c

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

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 int device_release(struct inode *, struct file *);
```

Add being_used In device_open in hello.c

```c
static int device_open(struct inode *inodep, struct file *filep){
        if(being_used){
                printk(KERN_ALERT "I'm being used!\n");
                return -EBUSY;
        }
        being_used++;
        timesCalled++;
        printk(KERN_INFO "Bye! :( BTW I've been called %d times\n", timesCalled);
        return 0;
}
```

Add being_used In device_release in hello.c

```
static int device_release(struct inode *inodep, struct file *filep){
        being_used--;
        printk("I'll never let go, Jack. I'll never let go. I promise.\n");
        return 0;
}
```

Let’s Also Fix That String Saving Bug

```
static ssize_t device_write(struct file *filep, const char __user *buf, size_t
length, loff_t *offset){
        long error_count;
        printk("Running device_write\n");
        memset(message,0,sizeof message);
        error_count = copy_from_user(message,buf,length);
        size_of_message = strlen(message);
        printk(KERN_INFO "Received %d characters from user\n",size_of_message);
        printk(KERN_INFO "Message received: %s\n", message);
        return length;
}
```

Also Update User File To Detect Error

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
static char receive[256];
int main(){
        int fd;
        int ret;
        char stringToSend[256];
        printf("Warm it up.exe\n");
        fd = open("/dev/meschar", O_RDWR);      // Capital o, not zero
        printf("Open return value: %d\n",fd);   // optional
        if(fd<0) return 0;
        printf("Do you know anything about the Chamber of Secrets?\n");
```

Run make
Test It Out

Now for The Mutex Approach

Add mutex header in driver file (hello.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>
#include <linux/mutex.h>

#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"
```

Add mutex header

```
#define DEVICE_NAME "meschar"
#define CLASS_NAME "mes"

static DEFINE_MUTEX(meschar_mutex);

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

Initialize Lock

```
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\n", multiplier);
        mutex_init(&meschar_mutex);
        return 0;
}
```

Check Lock

```c
static int device_open(struct inode *inodep, struct file *filep){
        if(!mutex_trylock(&meschar_mutex)){
                printk(KERN_ALERT "I'm being used!\n");
                return -EBUSY;
        }
        //being_used++;
        timesCalled++;
        printk(KERN_INFO "Bye! :( BTW I've been called %d times\n", timesCalled);
        return 0;
}
```

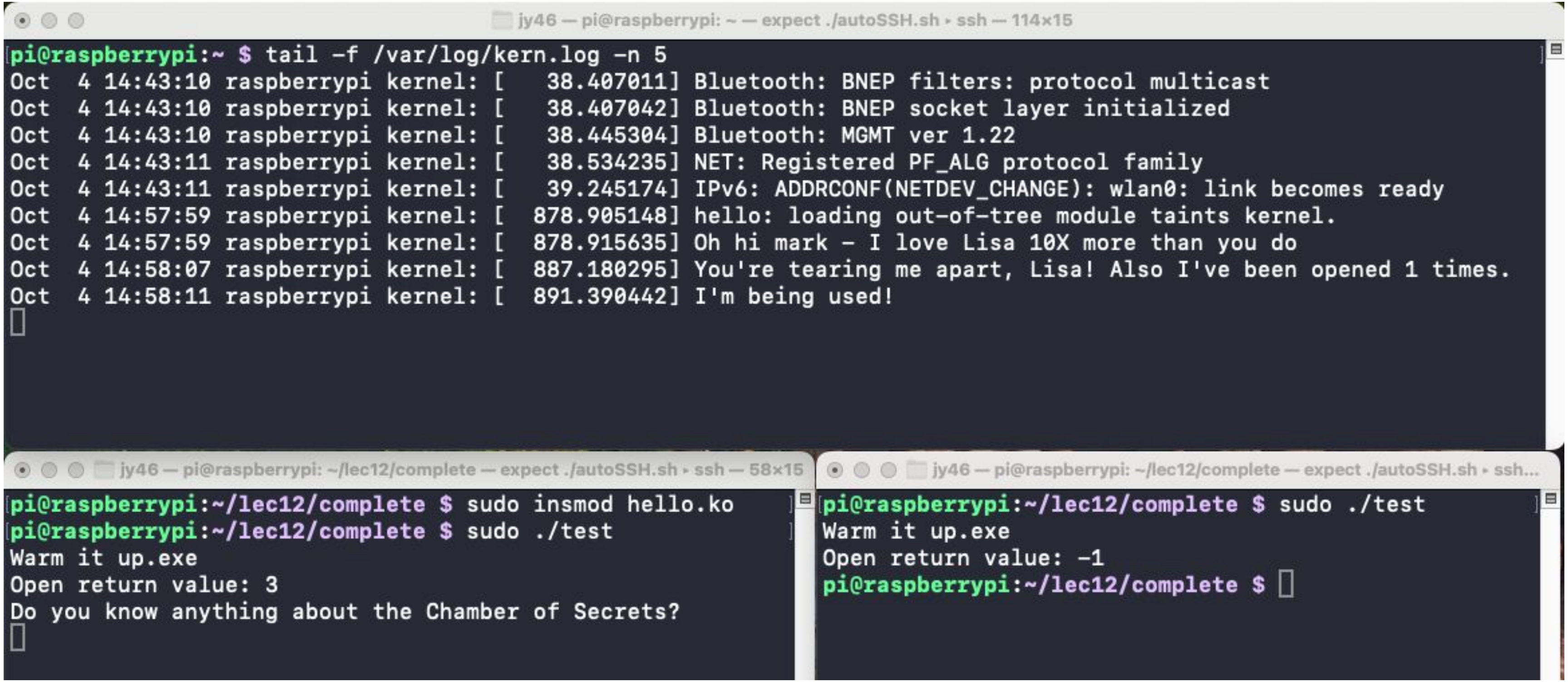
Release Lock

```
static int device_release(struct inode *inodep, struct file *filep){
        mutex_unlock(&meschar_mutex);
        //being_used--;
        printk("I'll never let go, Jack. I'll never let go. I promise.\n");
        return 0;
}
```

Destroy Lock

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

Run make
Then Test It Out



jy46 — pi@raspberrypi: ~ — expect ./autoSSH.sh ▸ ssh — 114×15
jy46 — pi@raspberrypi: ~/lec12/complete — expect ./autoSSH.sh ▸ ssh — 58×15
jy46 — pi@raspberrypi: ~/lec12/complete — expect ./autoSSH.sh ▸ ssh...

```
pi@raspberrypi:~ $ tail -f /var/log/kern.log -n 5
Oct  4 14:43:10 raspberrypi kernel: [   38.407011] Bluetooth: BNEP filters: protocol multicast
Oct  4 14:43:10 raspberrypi kernel: [   38.407042] Bluetooth: BNEP socket layer initialized
Oct  4 14:43:10 raspberrypi kernel: [   38.445304] Bluetooth: MGMT ver 1.22
Oct  4 14:43:11 raspberrypi kernel: [   38.534235] NET: Registered PF_ALG protocol family
Oct  4 14:43:11 raspberrypi kernel: [   39.245174] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
Oct  4 14:57:59 raspberrypi kernel: [  878.905148] hello: loading out-of-tree module taints kernel.
Oct  4 14:57:59 raspberrypi kernel: [  878.915635] Oh hi mark - I love Lisa 10X more than you do
Oct  4 14:58:07 raspberrypi kernel: [  887.180295] You're tearing me apart, Lisa! Also I've been opened 1 times.
Oct  4 14:58:11 raspberrypi kernel: [  891.390442] I'm being used!
```

```
pi@raspberrypi:~/lec12/complete $ sudo insmod hello.ko
pi@raspberrypi:~/lec12/complete $ sudo ./test
Warm it up.exe
Open return value: 3
Do you know anything about the Chamber of Secrets?
```

```
pi@raspberrypi:~/lec12/complete $ sudo ./test
Warm it up.exe
Open return value: -1
pi@raspberrypi:~/lec12/complete $ 
```

Submit a screenshot of the three terminal windows demonstrating that you have implemented mutex successfully. I have attached an example shot from me below. The three windows should be as follows:

Terminal 1: Displaying output of the kernel log (tail -f /var/log/kern.log -n 5), which shows module insertion/init message, open message, and the being used message.

Terminal 2: Showing that you ran sudo insmod hello.ko and then sudo ./test (don't type anything in response to the prompt, leave it there)

Terminal 3: Showing that you tried running sudo ./test again but were not able to complete the program (since terminal 2 is currently using it).