While You're Waiting - Update/Upgrade/Install

- SSH into your Pi
- Update and upgrade Raspberry Pi OS (*the OS formerly known as Raspbian*)
  - `sudo apt update && sudo apt -y upgrade`
  - The `-y` flag automatically answers y to the y/n prompt that would normally appear
- For fun, try to print system information (command: uname), specifically the kernel release (-r):
  - `uname -r`
- Install Linux kernel headers
  - `sudo apt-get install raspberrypi-kernel-headers`
- You may have to open a second terminal to write code while we wait for this to install
- Where are these headers installed?
  - `less /usr/src/linux-headers-6.1.21+/include/linux/init.h`
  - `less /usr/src/linux-headers-6.1.21+/include/linux/module.h`
  - q key exits less

ELEC 424/553

Mobile & Embedded Systems

Lecture 9 - Writing Your Own Module

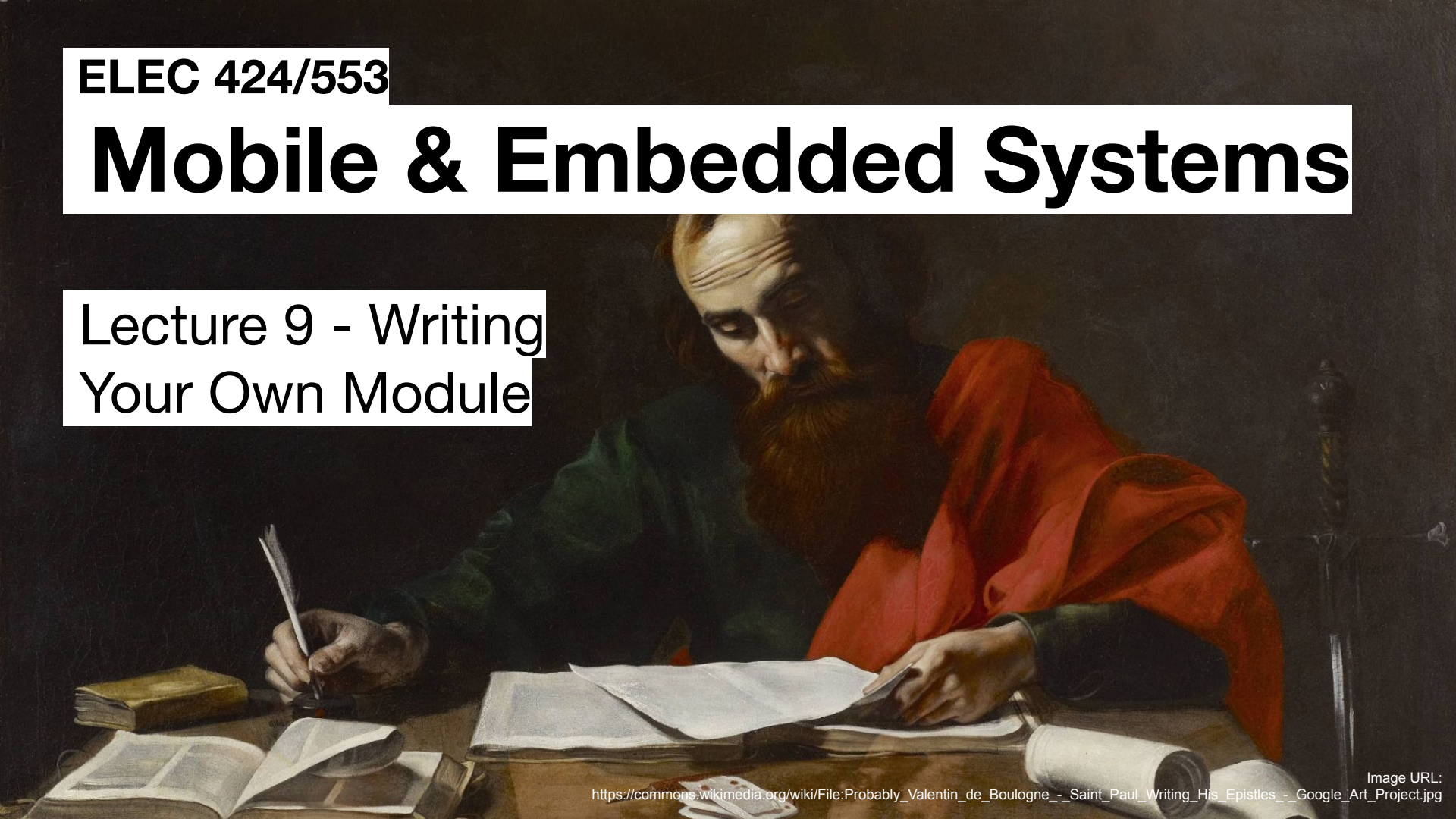


Image URL:
https://commons.wikimedia.org/wiki/File:Probably_Valentin_de_Boulogne_-_Saint_Paul_Writing_His_Epistles_-_Google_Art_Project.jpg

Don’t Do This At Home!

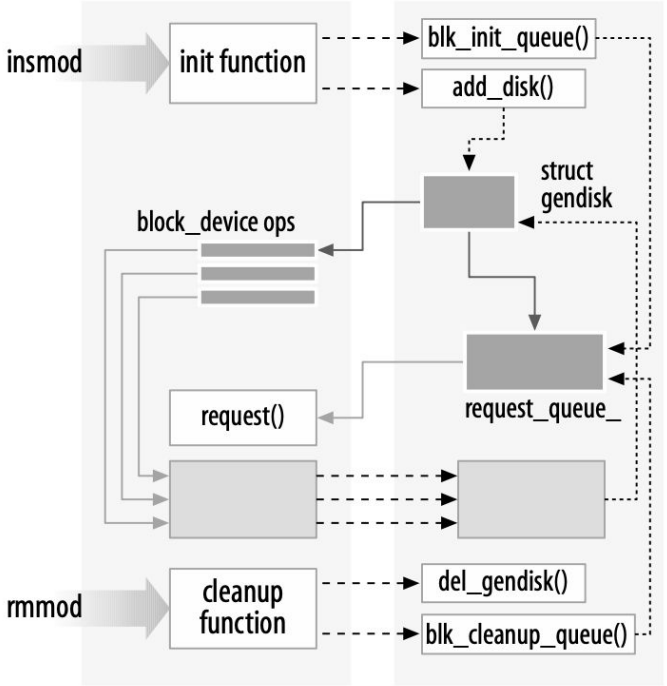
- Please do not using your primary laptop and operating system
  - However, you are unlikely to encounter issues
  - Just be sure to save anything else running on your computer and be ready for a random crash; Don’t hold me liable for this!
- Reportedly does not work on WSL, and will not work on CLEAR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

Update/Upgrade/Install

- Update and upgrade Raspberry Pi OS (*the OS formerly known as Raspbian*)
  - `sudo apt update && sudo apt -y upgrade`
  - The `-y` flag automatically answers y to the y/n prompt that would normally appear
- For fun, try to print system information (command: uname), specifically the kernel release (-r):
  - `uname -r`
- Install Linux kernel headers
  - `sudo apt-get install raspberrypi-kernel-headers`
- You may have to open a second terminal to write code while we wait for this to install
- Where are these headers installed?
  - `less /usr/src/linux-headers-6.1.21+/include/linux/init.h`
  - `less /usr/src/linux-headers-6.1.21+/include/linux/module.h`
  - q key exits less

Linux Kernel Modules (LKMs)

- **Loadable** object code
- Enable kernel to do more (or less) on the fly
- Module **classes**
    - E.g. **device drivers, file systems**
- Dynamically linked to kernel via **`insmod`**
- Dynamically unlinked to kernel via **`rmmod`**



Corbet, Rubini, & Kroah-Hartman, *Linux Device Drivers*, 3rd Ed. Fig. from p. 19 - full figure not shown.

The Code You'll See 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**

Let Me Get Started Before We Get Started

```
make
ls
sudo insmod hello.ko
dmesg
ls -l /sys/module/
ls /sys/module/hello/
sudo rmmod hello.ko
```

Download Makefile from the Files folder on Canvas

scp Makefile pi@raspberrypi.local:~/

```
# From: Dr. Derek Molloy, School of Electronic Engineering, Dublin City
# University, Ireland. URL:
# http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

obj-m+=hello.o   Goal definition: Object, module, adding, uses hello.c

        (Default) Target [Below: -C specifies directory for kbuild/Makefile
all:                              M specifies module source directory]
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:   Target
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

Note: kbuild's Makefile has modules and clean as targets

Now Make The World's Simplest Module: Oh Hi Mark
To do this, you could use (the file must be called hello.c given our Makefile): `nano hello.c`

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("GPL");
int hello_init(void) {
    printk("Oh hi mark\n");
    return 0;
}
void hello_exit(void){

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

Called upon module **loading**

We aren't in user space where we would have **`glibc`** and **`stdio`**; **So we must use printk**

Called upon module **removal**

Macros

Modified from: Corbet, Rubini, & Kroah-Hartman, *Linux Device Drivers*, 3rd Ed. URL: https://lwn.net/Kernel/LDD3/

Try it out!

```
make
ls
sudo insmod hello.ko
dmesg
ls -l /sys/module/
ls /sys/module/hello/
sudo rmmod hello.ko
```

From GIF from user ULTiMO_H3RO on tenor. URL:
https://tenor.com/view/e3-keanu-reeves-cyberpunk-check-this-out-gif-14474824

Adding On To The World's Simplest Module

```
#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");

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

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

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

Macros (link)

__init and __exit macros
Memory management for built-in drivers, which remove these functions when appropriate; Loadable modules keep them

We Should Make The Functions `static` [limit scope]

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

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

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

Add Log Level (KERN_INFO - log level 6) to printk()

```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 __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);
```

Try it out!

```
make
ls
sudo insmod hello.ko
dmesg
sudo rmmod hello.ko
dmesg
tail -f /var/log/kern.log
```

From GIF from user ULTiMO_H3RO on tenor. URL:
https://tenor.com/view/e3-keanu-reeves-cyberpunk-check-this-out-gif-14474824

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

Note - There Are Other Types of Module Parameters Besides `int`

- **`bool`**
- **`charp`** - char pointer
- **`long`**
- **`short`**
- **`uint`**
- …

Corbet, Rubini, & Kroah-Hartman, *Linux Device Drivers*, 3rd Ed. URL: https://lwn.net/Kernel/LDD3/

Permissions Mask - A disaster in terms of readability

- 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

What Does Linus Think?

On Tue, Aug 2, 2016 at 1:42 PM, Pavel Machek <pavel@ucw.cz> wrote:
>
> Everyone knows what 0644 is, but noone can read S_IRUSR | S_IWUSR |
> S_IRCRP | S_IROTH (*). Please don't do this.

Absolutely. It's *much* easier to parse and understand the octal numbers, while the symbolic macro names are just random line noise and hard as hell to understand. You really have to think about it.

So we should rather go the other way: convert existing bad symbolic permission bit macro use to just use the octal numbers.

The symbolic names are good for the *other* bits (ie sticky bit, and the inode mode _type_ numbers etc), but for the permission bits, the symbolic names are just insane crap. Nobody sane should ever use them. Not in the kernel, not in user space.

Posted August 3, 2016 by corbet, “Please don't replace numeric parameter like 0444 with macro”. URL: https://lwn.net/Articles/696229/

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);
```

Reminder: Our Makefile (reprinted below) will be used when we enter “make”

```
# From: Dr. Derek Molloy, School of Electronic Engineering, Dublin City
# University, Ireland. URL:
# http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

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=8

ls /sys/module/

ls /sys/module/hello/

ls /sys/module/hello/parameters/

cat /sys/module/hello/parameters/multiplier

dmesg

sudo rmmod hello.ko
```