Exercise 7

Your goal: Add code to print both printk statements the number of times given by multiplier

The following slides are from the lecture to help you with the code in case you didn't get everything down

Submit to Canvas:
1) Your hello.c (module) file
2) A screenshot showing your terminal window containing "sudo insmod hello.ko multiplier=X" [X being 3 to 4, your choice] and also containing the output of "tail -f /var/log/kern.log" which will prove that your module code works

To move the code back to your laptop:
1) Exit out of SSH
2) scp pi@raspberrypi.local:~/hello.c ~/
Note: You can put wherever you want in place of ~/, I just chose that for my destination. You could also just copy the code from terminal and paste the code into a new file on your computer.

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**

Add Log Level (KERN_INFO) to 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

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

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);
```

Your goal:
Add code to print both printk statements the number of times given by multiplier

Try it out!

```
sudo insmod hello.ko multiplier=100

ls /sys/module/

ls /sys/module/hello/

ls /sys/module/hello/parameters/

cat /sys/module/hello/parameters/multiplier

sudo rmmod hello.ko
```