

Mauro Lopez
LAB 5 Report
ECEN-449
Section 502
October 5, 2018

Introduction:

In this lab we will work on cross compiling a "Hello world" program on our kernel module located on the CentOS 7 workstations.

Procedure:

We start by creating a new directory and name it lab5. We also launch Linux on our ZYBO board, and use PICOCOM to interact with it.

We will first mount the SD card to our home directory. Some warning will show up, but we ignore this.

We type the following command:

- Mount /dev/mmcbk0p1 /mnt/

To test this command we open the new directory and typing the following commands:

- Cd /mnt/
- Ls -la

We will get the following from our terminal.

```
mauro31@lin08-424cvlb:~  
File Edit View Search Terminal Help  
dev licenses mnt root tmp  
etc linuxrc opt sbin usr  
zynq> cd mnt  
zynq> ls  
BOOT.bin          devicetree.dtb      uImage             uramdisk.image.gz  
zynq> run fsck  
-/bin/ash: run: not found  
zynq> fsck  
fsck (busybox 1.18.4, 2012-01-09 15:03:52 PST)  
zynq> ls  
BOOT.bin          devicetree.dtb      uImage             uramdisk.image.gz  
zynq> ls -la  
total 9469  
drwxr-xr-x   6 root    0                4096 Jan  1 00:00 .  
drwxr-xr-x  17 12319  300            1024 Jan  1 00:03 ..  
drwxr-xr-x   4 root    0                4096 Oct  5 2018 .Trash-324003313  
drwxr-xr-x   4 root    0                4096 Oct  5 2018 .Trash-326005545  
drwxr-xr-x   4 root    0                4096 Oct  5 2018 .Trash-725002743  
drwxr-xr-x   4 root    0                4096 Oct  5 2018 .Trash-924006772  
-rwxr-xr-x   1 root    0           2522540 Oct  5 2018 BOOT.bin  
-rwxr-xr-x   1 root    0              7446 Sep 28 2018 devicetree.dtb  
-rwxr-xr-x   1 root    0          3447856 Oct  5 2018 uImage  
-rwxr-xr-x   1 root    0          3693174 Sep 29 2018 uramdisk.image.gz  
zynq>
```

We will now test the permission in our SD card. We will try to create a directory in it.

We use the following commands:

- Mkdir test
- ls -lae

to avoid corruption of our files we need to type the following commands.

- cd /
- umount /mnt

If we go back to our “mnt” directory we will see that it is now empty. This will help prevent our files from being corrupted.

Now that we have access to read/write into our sd card, we can launch our first program call “hello” world .

```

1.  /* hello.c - hello world kernel module
2.  * Demonstrates module initialization, module release and printk
3.  *
4.  * (Adapted from various example modules including those found in the
5.  * Linux Kernel programming guide, Linux Device Drivers book and
6.  * FSM's device driver tutorial )
7.  */
8.
9.
10. #include <linux/module.h>
11. #include <linux/kernel.h>
12. #include <linux/init.h>
13.
14. /* This function is run upon module load. this is where you setup data
15.  * structures and reserve resources used by the module
16.  */
17. static int __init my_init(void){
18.     /*Linux kernel's version printf */
19.     printk(KERN_INFO "Mauro Lopez\n hello world! \n ");
20.     return 0;
21. }
22.
23. /* this function is run just prior to the module's removal from the system.
24.  you should release _ALL_ resources used by your module
25.  * here (otherwise be prepared for a reboot).
26.  */
27. static void __exit my_exit(void){
28.     printk(KERN_ALERT "Goodbye world! \n");
29. }
30. /* these define info that can be displayed by modinfo */
31. MODULE_LICENSE("GPL");
32. MODULE_AUTHOR("ECEN449 Student (and Others)");
33. MODULE_DESCRIPTION("Simple Hello World Module");
34. /* here we define which functions we want to use for initialization
35.  and clean up */
36. module_init(my_init);
37. module_exit(my_exit);

```

once we have type the code above into our “hello.c”, we need to make a makefile to compile the code.

We do this by doing the following :

```

1. obj-m += hello.o
2.
3. all:
4.     make -C /home/ugrads/m/mauro31/fall18/ecen449/lab4/linux-3.14 M=$(PWD) modules
5.
6.
7. clean:
8.     make -C /home/ugrads/m/mauro31/fall18/ecen449/lab4/linux-3.14 M=$(PWD) clean

```

Once we have our make file, we can use it to compile our c code. We do this by typing the following command:

➤ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-

If our code compiles, we will end up with a hello.ko file. in our modules file.

Now we upload our file into, our SD card, and placed it back to our board.

From our home in our ZYBO board we need to mount the sd card once again, to access our hello.ko file.

So once again we type the following commands.

➤ mount /dev/mmcblk0p1 /mnt/

now we can load our module into our board. We do this by using the following commands:

➤ insmod hello.ko

we should get the following :

```

mauro31@lin08-424cvlb:~
File Edit View Search Terminal Help
dev licenses mnt root tmp
etc linuxrc opt sbin usr
zynq> cd mnt
zynq> ls
BOOT.bin          hello.ko          uImage
devicetree.dtb    test             uramdisk.image.gz
zynq> insmod hello.ko
Mauro Lopez
hello world!
zynq> dmesg | tail
RAMDISK: gzip image found at block 0
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SL08G 7.40 GiB
mmcblk0: p1
EXT2-fs (ram0): warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing unused kernel memory: 212K (40627000 - 4065c000)
random: dropbear urandom read with 1 bits of entropy available
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
Please run fsck.
zynq> lsmod
hello 566 0 - Live 0x3f000000 (0)
zynq>

```

```
mauro31@lin08-424cvlb:~
File Edit View Search Terminal Help
EXT2-fs (ram0): warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing unused kernel memory: 212K (40627000 - 4065c000)
random: dropbear urandom read with 1 bits of entropy available
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
Please run fsck.
zynq> lsmod
hello 566 0 - Live 0x3f000000 (0)
zynq> mkdir -p /lib/modules/
zynq> mkdir -p /lib/modules/`uname -r`
zynq> ls
BOOT.bin          hello.ko          uramdisk.image.gz
devicetree.dtb    uImage
zynq> ls
BOOT.bin          hello.ko          uramdisk.image.gz
devicetree.dtb    uImage
zynq> rmmod hello

Goodbye world!
zynq> ls
BOOT.bin          hello.ko          uramdisk.image.gz
devicetree.dtb    uImage
zynq> █
```

Compile a kernel module that reads and write to the multiplication peripheral :

To run this program, we will do similar things to our “hello.world”. The only difference being, that we will use our multiplication ip from our previous lab.

We start by creating a lab5b directory to store a copy of our modules.

To make this program to run, we are going to need our “.h” files such as ; “xparameter.h”, and “xparameter_pps.h” to be in our module directory.

We can now create our multiply.c file, it should look something like this:

```
1. #include <linux/module.h> /* needed by all modules */
2. #include <linux/kernel.h> /* needed for KERN_ and print k*/
3. #include <linux/init.h> /* needed for __init and __exit macros*/
4. #include <asm/io.h> /* needed for IO reads and writes */
5.
6. #include "xparameters.h" /* needed for physical address of multiplier*/
7.
8. #define PHY_ADDR XPAR_MULTIPLY_0_S00_AXI_BASEADDR /* Physical address of multiplier */
9. #define MEMSIZE XPAR_MULTIPLY_0_S00_AXI_HIGHADDR - XPAR_MULTIPLY_0_S00_AXI_BASEADDR+1
10.
11. void * virt_addr;
12.
```

```

13. static int __init my_init(void)
14. {
15.
16.
17. printk(KERN_INFO "Mapping virtual address..\n");
18. virt_addr=ioremap(PHY_ADDR,MEMSIZE); // mapping our address.
19. //writing in our registers
20. printk(KERN_INFO "Writing a 7 to the register 0\n");
21. iowrite32( 7, virt_addr+0);
22. // writing into our registers
23. printk(KERN_INFO "Writing a 2 to the register 1\n");
24. iowrite32( 2, virt_addr+4);
25. //will print our the result of our multiplication
26. printk("read %d from register 0\n", ioread32(virt_addr+0));
27. printk("read %d from register 1\n", ioread32(virt_addr+4));
28. printk("read %d from register 2\n", ioread32(virt_addr+8));
29.
30. return 0;
31. }
32. /*
33. this function run just prior to the modules removal from the system.
34. * you should release _all_ resources used by your module
35. * here (otherwise be prepared for a reboot) .
36. */
37. static void __exit my_exit(void)
38. {
39. printk(KERN_ALERT "unmapping virtual address space.... \n");
40. iounmap( (void*)virt_addr);
41.
42. }
43.
44. // info that can be displayed by modinfo
45. MODULE_LICENSE("GPL");
46. MODULE_AUTHOR("ECEN449 Student (and others)");
47. MODULE_DESCRIPTION("Simple multiplier module");
48. /* Here we define which functions we want to use for initialization
49. * and cleanup
50. */
51. module_init(my_init);
52. module_exit(my_exit);

```

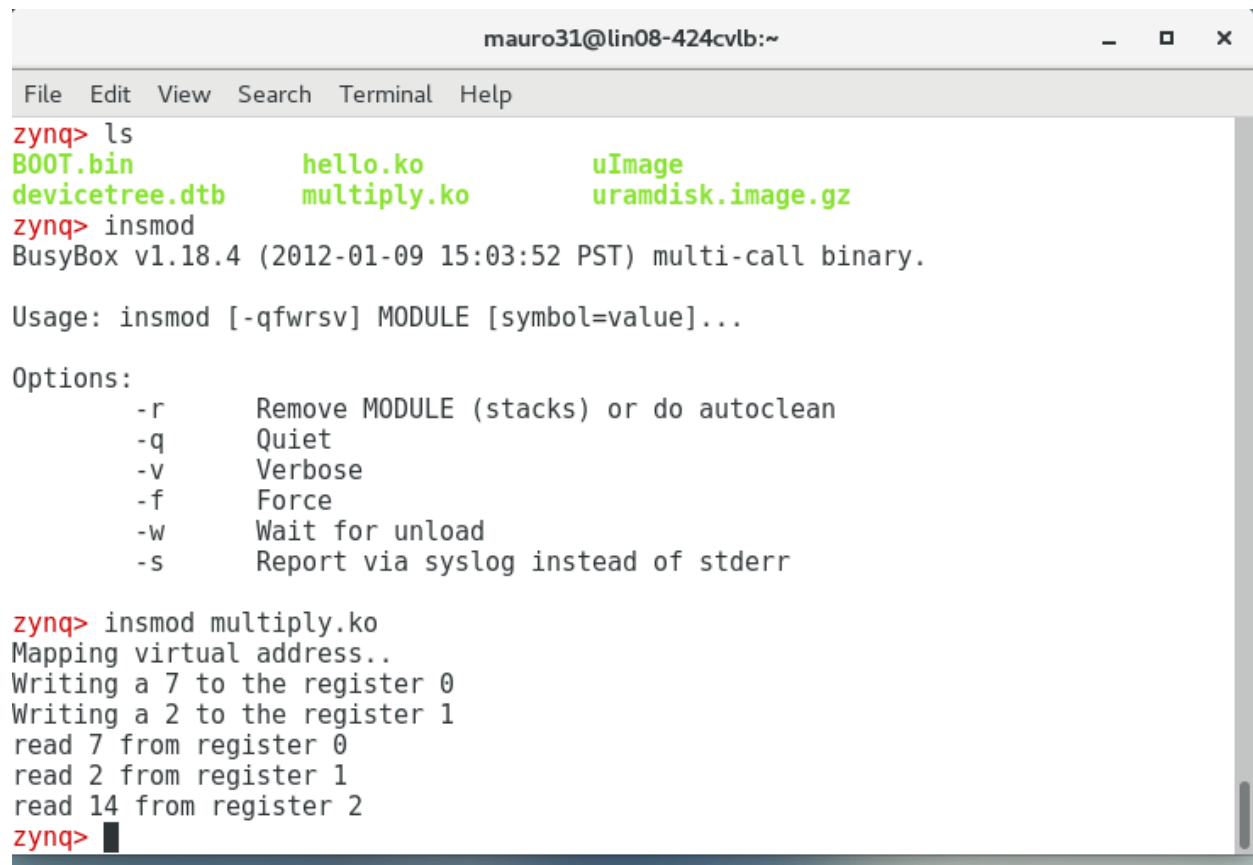
once again, we need to make our make file to compile our c code. We repeat the similar steps as we did before and we get the following.

```

1. obj-m += multiply.o
2.
3. all:
4.     make -C /home/ugrads/m/mauro31/fall118/ecen449/lab4/linux-3.14 M=$(PWD) modules
5.
6.
7. clean:
8.     make -C /home/ugrads/m/mauro31/fall118/ecen449/lab4/linux-3.14 M=$(PWD) clean

```

We should get the following :

A terminal window titled 'mauro31@lin08-424cvlb:~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
zynq> ls
BOOT.bin          hello.ko          uImage
devicetree.dtb    multiply.ko       uramdisk.image.gz
zynq> insmod
BusyBox v1.18.4 (2012-01-09 15:03:52 PST) multi-call binary.

Usage: insmod [-qfwrsv] MODULE [symbol=value]...

Options:
  -r      Remove MODULE (stacks) or do autoclean
  -q      Quiet
  -v      Verbose
  -f      Force
  -w      Wait for unload
  -s      Report via syslog instead of stderr

zynq> insmod multiply.ko
Mapping virtual address..
Writing a 7 to the register 0
Writing a 2 to the register 1
read 7 from register 0
read 2 from register 1
read 14 from register 2
zynq> █
```

Result:

From the screenshots above, I was able to complete the lab requirements. I was able to run the Hello.ko and multiply.ko programs. I was having issues with the first lab, given that it wasn't compiling because the way the file was formatted wasn't correct. Once I got it to work, I was able to use what I learned to solve for the multiply.o . I notice that our multiply.ko is very similar to our multiply.c from our previous labs.

Conclusion:

In This lab I was able to cross compile a "hello world" and "multiply" program in c, and run it directly from my ZYBO board. I learned more about creating Kernel modules, and how to run them in my ZYBO board. The lab helped me have a better understanding of the linux operating system. I was able to use commands such as "lsmod, rmod" to debug, my modules in case something wasn't working. Overall the lab, was a success.

Questions:

A) If prior to step 2.f, we accidentally reset the ZYBO board, what additional steps would be needed in step 2.g?

We would have to relaunch our PICOCOM, once again. We would need to mount our sd card. Assuming our cards doesn't corrupt, we would have to format it again.

B) What is the mount point for the SD card on the CentOS machine? Hint: Where does the SD card lie in the directory structure of the CentOS file system.

The SD card lies in the directory named: /run/media/mauro31

C) If we changed the name of our hello.c file, what would we have to change in the Makefile? Likewise, if in our Makefile, we specified the kernel directory from lab 4 rather than lab 5, what might be the consequences?

We would have to change the following line :

"obj-m += hello.o " to the new file name.

Since we are still using the same tools to compile, we wont see a change, as long as we have the right directories.