Lab #5:

Simple Kernel Module

Khanh Nguyen

UIN# 525000335

ECEN 449– 503

Date: February 23, 2017

INTRODUCTION:

The purpose of this lab is to learn how to create and cross compile simple “Hello World!” and multiply modules and load them into Linux kernel on the ZYBO board.

PROCEDURE:

1/ Load PICOCOM serial and boot Linux using ZYBO board.

2/ Test the mount, write commands and then unmount the SD card.

3/ Copy over the contents of lab 4 and create a folder call “modules” under lab 5 directory.

4/ Create a hello.c file with the provided code under ‘modules’ folder.

5/ Create a Makefile file with the provided code.

6/ Cross-compile the hello.c module.

7/ Copy the generated hello.ko into the SD card.

8/ Mount the SD into the ZYBO board and load the module into Linux kernel.

9/ Creating kernel image by running cross compiler linux configuration for ARM processor.

10/ Create lab5b directory and copy over the “modules” folder.

11/ Create a new module called “multiply.c”.

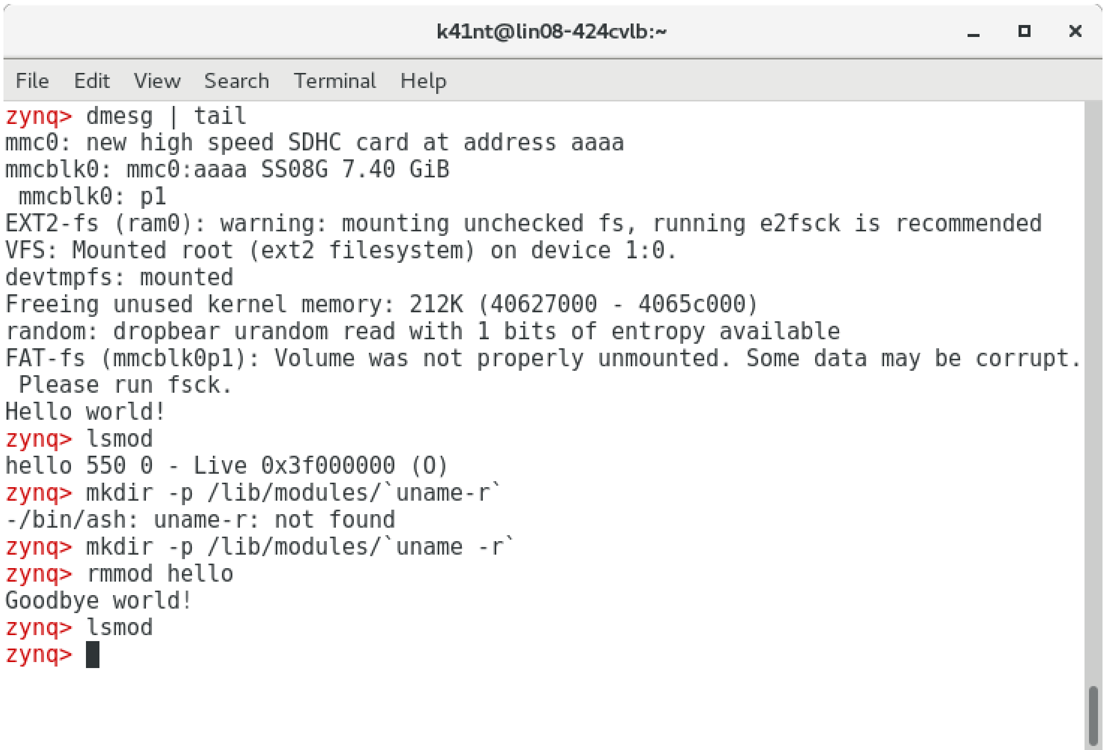
12/ Modify the code provided.

13/ Copy the xparameters.h and x_parameters_ps.h into “modules” directory .

14/ Open picocom terminal, copy BOOT.bin, uImage, uramdisk.image.gz and devicetree.dtb on to SD card then plug it into the Zybo board.

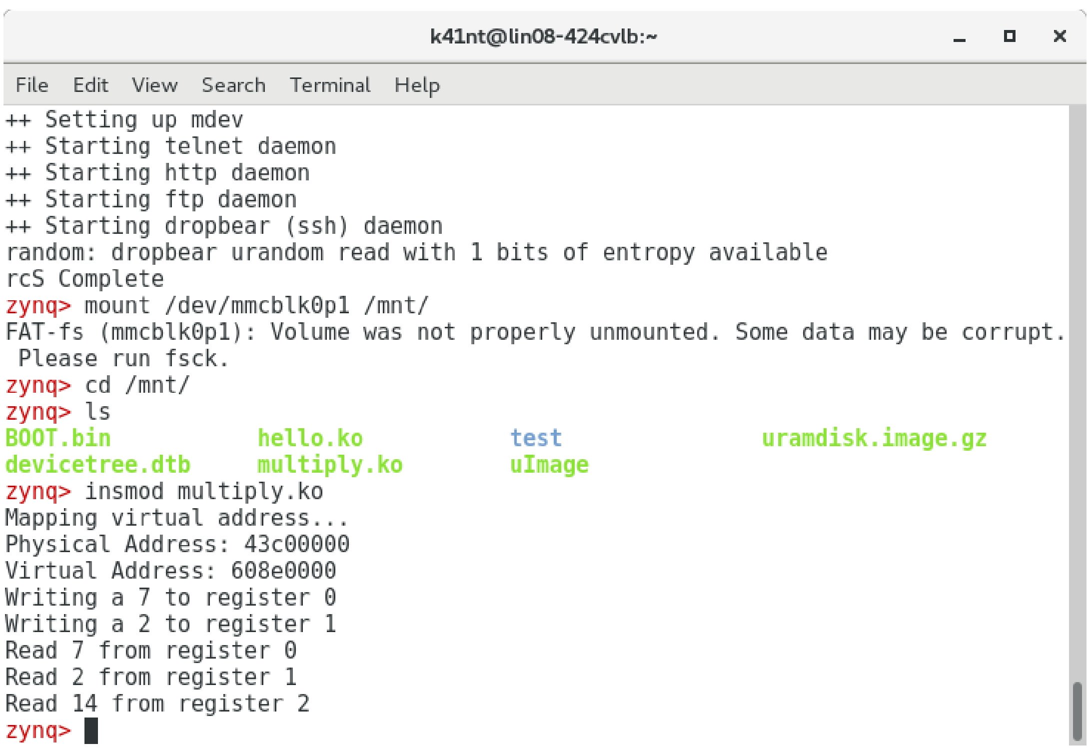
15/ Cross compile the module and mount the SD cards like previous steps with hello.c module.

RESULT:



```
k41nt@lin08-424cvlb:~
File Edit View Search Terminal Help
zynq> dmesg | tail
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SS08G 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.
Hello world!
zynq> lsmod
hello 550 0 - Live 0x3f000000 (O)
zynq> mkdir -p /lib/modules/`uname-r`
-/bin/ash: uname-r: not found
zynq> mkdir -p /lib/modules/`uname -r`
zynq> rmmod hello
Goodbye world!
zynq> lsmod
zynq>
```

Result screenshot of “Hello World!” module



```
k41nt@lin08-424cvlb:~
File Edit View Search Terminal Help
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
random: dropbear urandom read with 1 bits of entropy available
rcS Complete
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
 Please run fsck.
zynq> cd /mnt/
zynq> ls
BOOT.bin          hello.ko          test              uramdisk.image.gz
devicetree.dtb    multiply.ko       uImage
zynq> insmod multiply.ko
Mapping virtual address...
Physical Address: 43c00000
Virtual Address: 608e0000
Writing a 7 to register 0
Writing a 2 to register 1
Read 7 from register 0
Read 2 from register 1
Read 14 from register 2
zynq>
```

Result screenshot of multiply module

C Code:

Multiply module:

```
#include <linux/module.h> /* Needed by all modules */

#include <linux/kernel.h> /* Needed for KERN_* and printk */

#include <linux/init.h> /* Needed for __init and __exit macros */

#include <asm/io.h> /* Needed for IO reads and writes */


#include "xparameters.h" /* Needed for physical address of multiplier */


/*from xparameters.h*/

#define PHY_ADDR XPAR_MULTIPLY_0_S00_AXI_BASEADDR //physical address of multiplier

/*size of physical address range for multiple */

#define MEMSIZE XPAR_MULTIPLY_0_S00_AXI_HIGHADDR - XPAR_MULTIPLY_0_S00_AXI_BASEADDR+1


void* virt_addr; //virtual address pointing to multiplier


/* This function is run upon module load. This is where you setup data structures and reserve resources used by the module. */


static int __init my_init(void) {

	/* Linux kernel's version of printf */

	printk(KERN_INFO "Mapping virtual address...\n");


	/*map virtual address to multiplier physical address*/

	//use ioremap

	virt_addr = ioremap(PHY_ADDR, MEMSIZE);
```

```c
	/*write 7 to register 0 */
	printk(KERN_INFO "Writing a 7 to register 0\n");
	iowrite32(7, virt_addr+0); //base address + offset
	/* Write 2 to register 1*/
	printk(KERN_INFO "Writing a 2 to register 1\n");
	//use iowrite32
	iowrite32(2, virt_addr+4); //base address + offset

	printk("Read %d from register 0\n", ioread32(virt_addr+0));
	printk("Read %d from register 1\n", ioread32(virt_addr+4));
	printk("Read %d from register 2\n", ioread32(virt_addr+8));
	printk("Physical Address: %x\n", PHY_ADDR); //Print physical address
	printk("Virtual Address: %x\n",*(int*)virt_addr); //Print virtual address

	//a non 0 return means init_module failed; module can't be loaded.
	return 0;
}
/* This function is run just prior to the module's removal from the system. You should release
_ALL_ resources used by your module here (otherwise be prepared for a reboot). */
static void __exit my_exit(void) {
	printk(KERN_ALERT "unmapping virtual address space...\n");
	iounmap((void*)virt_addr);
}

/* These define info that can be displayed by modinfo */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Khanh Nguyen");
MODULE_DESCRIPTION("Simple multiplier module");
```

```
/* Here we define which functions we want to use for initialization and cleanup */
module_init(my_init);
module_exit(my_exit);
```

CONCLUSION:

The modules were compiled and loaded into the Linux kernel on ZYBO board successfully. I learned how create a module using C and makefile for the module. I was able to examine the physical memory and virtual memory addresses during the process of making the multiply.c module. I also learned how to use the 'ioremap' and 'iounmap' functions.

QUESTIONS:

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

We should use the SD card to reboot the Linux on ZYBO board and then mount the SD card.

2/ 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.

/run/media/k41nt/98DD-AC9A

3/ 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

If we changed the name of the hello.c to anything other name, we have to change hello.c in makefile to that name as well.

Using the lab 4 kernel directory may make our program not working properly.