Description: Description: Description: Description: Description: Description: Description: Description: Description: Description: Texas A&M University Logo - aTm

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”.

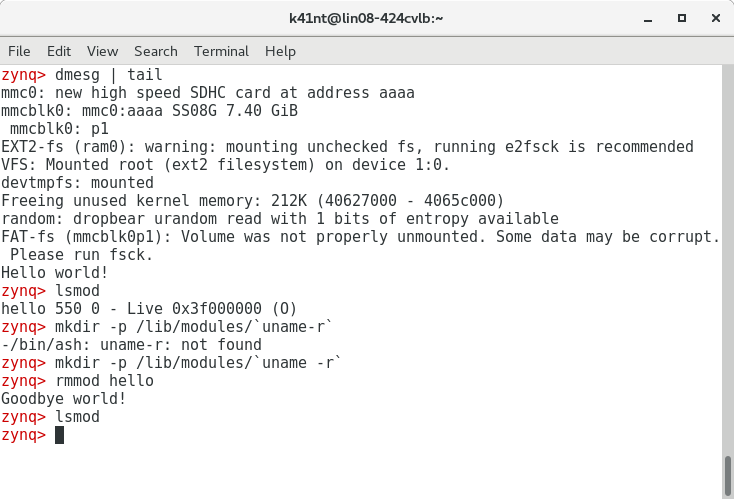
12/ Modify the code provided.

13/ Copy the xparameters.h and x\_parameters\_ps.h into “modules” directory .

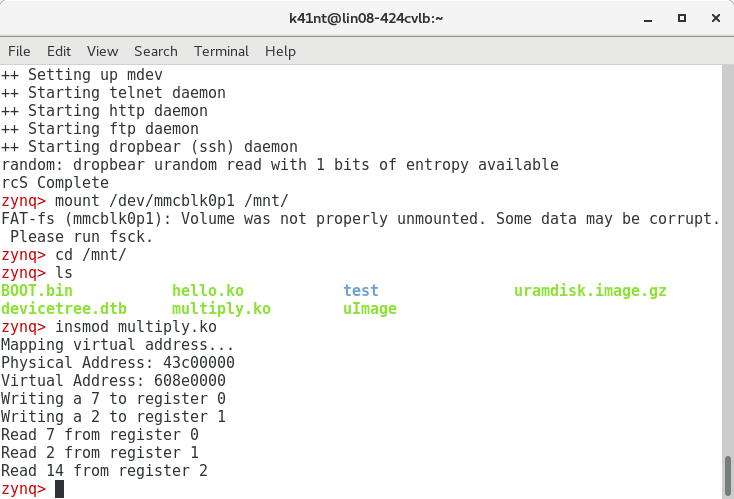
14/ Open picocom terminal, copy BOOT.bin, uImage, uramdisk.image.gz and devicetree.dtb on to SD card then plug it into the Zybo board.

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

**RESULT:**



Result screenshot of “Hello World!” module



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);

/\*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.