ECEN 749- Microprocessor System Design

Section 603

TA: Mr. Kunal Bharathi

LAB 5

Introduction to Kernel Modules on Zynq Linux System

Date of Performance: 10/03/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

Introduction

This lab focusses on creating linux kernel module and then loading the module into the linux kernel on the ZYBO Z7-10 board. It also focusses on the creating a kernel module to print the messages to the Kernel's message buffer and moderating kernel access to an existing peripheral.

Design:

Part 1.

This lab requires file from the previous lab and hence we need to copy those files in the current directory.

1. Boot Linux on ZYBO Z7-10 and then mount the SD card.
2. To mount the SD card, type the following command on the ZYBO Z7-10 linux serial console:
   >mount /dev/mmcblk0p1 /mnt/
3. After that, to test the mounting, run the following command.
   >cd /mnt/
   >ls -la
   The first command changes the current directory to the SD card mount point. The next command lists the contents of the current directory.
4. Run the following commands in PICOCOM:
   >mkdir test
   >ls -lae
5. The SD card is used to transfer the files from the PC (CentOS workstation) to the ZYBO Z7-10 board. Before removing the SD card from the ZYBO board, first unmount the FAT partition on the SD card.
   > cd /
   > umount /mnt
   The first command changes the current directory to the root directory and the second command unmounts the FAT partition.
6. Now change the current directory to '/mnt/' using 'cd'. If the contents of the directory is checked, you will see nothing as the SD card is already unmounted.

Part2: Cross-compiling Kernel module.

1. Remove the SD card without powering off the ZYBO Z7-10 and insert it into the SD card.
2. Create a directory under lab5 called 'modules' and copy the text below into a file called 'hello.c'

```c
/*  hello.c - Hello World kernel module
 *
 *  Demonstrates module initialization, module release and printk.
 *
 *  (Adapted from various example modules including those found in the
 *  Linux Kernel Programming Guide, Linux Device Drivers book and
 *  FSM's device driver tutorial)
 */

#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 */

/* 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 "Hello dhiraj!\n");    // REPLACE WITH YOUR NAME

  // 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 "Goodbye dhiraj!\n");
}

/* These define info that can be displayed by modinfo */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Student (and others)");
MODULE_DESCRIPTION("Simple Hello World Module");

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

3. Makefile is used to create the object file from the .c file. n your 'modules' directory, create a file called 'Makefile' and fill it with the text below where <kernel source directory> is the root directory of the Linux kernel source code for lab 5.

   obj-m+= hello.o

   all:

   make -C /home/grads/d/dhirajkudva/ecen749lab/Lab4/linux-3.14 M=$(PWD) modules

   clean:

   make -C /home/grads/d/dhirajkudva/ecen749lab/Lab4/linux-3.14 M=$(PWD) clean
4. To obtain the .ko file, write the following command
   >make ARCH=arm CROSS COMPILE=arm-xilinx-linux-gnueabi-
5. Copy the generated .ko file to SD card.
6. Now insert the SD card in Zybo board and execute the following command:
   >insmod hello.ko
7. To see the output of the 'printk' statement that is called when the kernel module is loaded run the following command in the terminal:
   >dmesg | tail
8. The current running modules can be seen using 'lsmod' in the terminal window.
9. Create a modules directory /lib/modules/`uname -r`using the following command
   mkdir -p /lib/modules/ `uname -r`
10. To remove the module, run 'rmmod hello', and then run 'lsmod' to ensure that the module was removed. Also run 'dmesg' again to examine the output of the module during removal.

Part 3: Compile a kernel module that reads and writes to the multiplication peripheral and prints the results to the kernel message buffer.

1. Create a new lab5 directory and copy the 'modules' directory.
2. Generate a makefile for multiply.o and ensure that it points to the linux kernel directory.
3. Create a new kernel module source file called 'multiply.c'
4. Copy the 'xparameters.h' and 'xparameters ps.h' files in to modules directory. These files should be located in your lab5/lab4.sdk/FSBL bsp folder.
5. Write the following code in multiply.c

```
#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 IO reads and writes
#include <linux/ioport.h>   // Used for io memory allocation

// From xparameters.h, physical address of multiplier
```

```
#define PHY_ADDR XPAR_MULTIPLY_0_S00_AXI_BASEADDR
// Size of physical address range for multiply
#define MEMSIZE XPAR_MULTIPLY_0_S00_AXI_HIGHADDR -
XPAR_MULTIPLY_0_S00_AXI_BASEADDR + 1

// virtual address pointing to multiplier
void* virt_addr;

/* 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

   virt_addr=ioremap(PHY_ADDR,MEMSIZE);
   printk("The physical address is %x",PHY_ADDR);
   printk("\n The VIRTUAL address is %x",virt_addr);
// 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");

    iowrite32(2, virt_addr + 4);

   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));

   // 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 Student (and others)");
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);
```

6. After this compile, the kernel file and then repeat the same steps for generating the.ko file. Load this .ko file in the SD card.
7. Mount the SD card on the ZYBO Z7-10 linux system using '/mnt' as the mount point.
8. Run insmod multiply.ko to load the kernel module and get the output.

Output:

When card is removed:

```
zynq> cd /
zynq> cd /mnt/
zynq> ls
zynq> mmc0: card aaaa removed
```

The output of part 2: hello.ko

```
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
 Please run fsck.
Hello dhiraj!
Goodbye dhiraj!
zynq>
```

The module running when hello.ko is executed.

```
zynq> lsmod
hello 550 0 - Live 0x3f038000 (O)
```

The output of the multiply.ko:

```
zynq> insmod multiply2.ko
Mapping virtual address...
The physical address is 43c00000
The VIRTUAL address is 60920000
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
```

Result:

The lab introduced to the concept of cross compiling, Because of this, we can run programs on different platforms and also load the program on FPGA. The main precaution that is to be taken while performing this experiment is that of unmounting the SD card. While removing from the ZYBO board we write the unmount command, and click on eject while removing from the Cent OS workstation. If this is not followed, then the SD card can get corrupted and we won't be able to access the files in the SD card.

Conclusion:

The lab helped in understanding the basics of creating a linux kernel module and using it for cross compiling over different platforms.

Questions:

a. If prior to step 2.f, we accidentally reset the ZYBO Z7-10 board, what additional steps would be needed in step 2.g?

Ans. If we accidently reset the ZYBO Z7-10 board before step 2.f, we need to mount the SD card and boot the linux OS on the ZYBO Z7-10 by pressing the Reset button. Since powering off would have vanished the read write permission, we need to restore the permissions again by using the following command:

>mount /dev/mmcb1k0p1 /mnt/

And then mounting operation of the SD card.

>cd /mnt/
>ls -la

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.?)

Ans. On the CentOS machine, the SD card lies in the /run/media/user. The SD card lies in the directory "/mnt/" directory of the linux user system.

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?

Ans. If we change the name of our hello.c file, then we have to provide that changed name in the makefile. If we did not write the new name of the file in the makefile, then it won't find the .c file and give us an error of "No rule to make target" and the .o and because of that .ko file won't be generated. If we specify the kernel directory from lab 4 instead of lab 5, there would be no change as all the resources needed for the makefile is already present in the lab 4 directory as well. In order to change the kernel directory back to its correct directory (in this case Lab 5 directory) , we need to execute -c command to change its directory and -f command to make the makefile follow the directory path. This can also be executed by a single line commad "cd/other/<lab5 directory> &&make". This will not change the shell's current directory, but will run make with the indicated working directory.