Kyle Loyka

Lab 5 Report

ECEN 449-503

Due: 29 February 2016

Introduction
The purpose of this lab is to expand on the knowledge from lab 4. In this lab, a kernel will be compiled that will support mounting non-volatile storage, and a kernel module will be loaded onto the FPGA board running Linux.

Procedure
In this lab, the kernel from lab 4 was recompiled with the necessary modules to read and write onto a FAT partition on a Compact Flash card.

In the General Setup, the initramfs source file was change from the *minimal* version to the *complete* version: `initramfs_complete.cpio.gz`. The kernel was also configured to support MSDOS and VFAT file systems. Additionally, some language support options had to be configured to support mounting these file systems.

The kernel was recompiled and loaded onto the FPGA board using the .ace file and CF card.

The second part of the lab involved writing a kernel module to display "Hello World". We learned about the kernel function `printk()` and macros `__init` and `__exit`. This module was compiled using the lab 5 kernel. The kernel module file (hello.ko) was loaded onto the CF card and mounted on the FPGA Linux system. After navigating to the mount directory and finding the hello.ko file, the kernel module was loaded into the Linux kernel.

Results / Q&A
The project functioned as intended.

(A) If prior to *step 3.f*, the XUP board was reset, the user would need to remake the mount directory before continuing to *step 3.g*.

(B) The mount point for the CF card on the CentOS machine is */media/TRANSCEND_*

(C) If the name of the hello.c file was changed, the name of hello.o in the makefile would also need to change to the new file name.

If in the makefile, the the lab 4 kernel directory was specified (instead of lab 5's kernel directory), the kernel module would be compiled based on "initramfs_minimal.cpio.gz" RAM system instead of lab 5's kernel module which is based on the "initramfs_complete.cpio.gz" RAM system. The minimal RAM system is only useful for compiling and running the basic kernel. The complete RAM system is important because it allows for the mounting of other devices. Using the lab 4 kernel directory (instead of lab 5) would result in a module that wouldn't work properly as it would be compiled for a different (and less advanced) kernel, which doesn't support mounting.

Conclusion

This lab provided a strong foundation for compiling and configuring operating systems to run on embedded devices. Notably, we learned how to get a kernel running on an FPGA board. We also learned about mounting devices and loading kernel modules on Linux based systems.

```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 kernel 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 World!\n");

 // 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 World!\n");
}

/* These define info that can be displayed by modinfo */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ECEN449 Student: Kyle Loyka");
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);
```

KERMIT OUTPUT (PART 3)

```
# mount -t vfat /dev/xsa1 /lib/modules/2.6.35.7
xsysace 83600000.sysace: capacity: 1957536 sectors
 xsa: xsa1
# cd /lib/modules/2.6.35.7/
# ls
hello.ko          linux_system.ace  test
# insmod hello.ko
Hello World!
# dmesg|tail
 xsa: xsa1
Xilinx SystemACE device driver, major=254
i2c /dev entries driver
Freeing unused kernel memory: 12187k freed
xsysace 83600000.sysace: No CF in slot
end_request: I/O error, dev xsa, sector 63
FAT: unable to read boot sector
xsysace 83600000.sysace: capacity: 1957536 sectors
 xsa: xsa1
Hello World!
# lsmod
hello 474 0 - Live 0xd0017000
# rmmod hello
Goodbye World!
# lsmod
# dmesg|tail
Xilinx SystemACE device driver, major=254
i2c /dev entries driver
Freeing unused kernel memory: 12187k freed
xsysace 83600000.sysace: No CF in slot
end_request: I/O error, dev xsa, sector 63
FAT: unable to read boot sector
xsysace 83600000.sysace: capacity: 1957536 sectors
 xsa: xsa1
Hello World!
Goodbye World!
```