ECEN 449: Microprocessor System Design
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Peng Li
TA: Andrew Targhetta
(Lab exercise created by A Targhetta and P Gratz)

Laboratory Exercise #5

'Hello World!' Kernel Module on XUP Linux System

Objective

The objective of this week's lab is to enable a non-volatile file system for data storage on the XUP Linux system built last week. This will be implemented via the CF card. In addition, you will cross compile a simple "Hello World!" kernel module on the CentOS workstations and load the module into the Linux kernel on the XUP board.

System Overview

The hardware system you will use this week is identical to the one built in Lab 4. The Linux kernel, however, will need to have a few more capabilities in order to access the FAT partition on the CF card. Likewise, a more complete pre-built root file system must be utilized to allow dynamic loading of kernel modules.

Procedure

1. The first part of lab involves recompiling the kernel you compiled last week to incorporate the necessary modules to read and write to the FAT partition on the CF card.

   (a) If you have not already done so, make a copy of your "lab4" directory and name it "lab5". This directory should include the Linux kernel source.

(b) For the last lab, we included a very minimalistic RAM file system. However, for this lab, we need to use some system utilities not included in the minimalistic variant so we will use a more complete RAM file system. Copy '/homes/faculty/shared/ECEN449/initramfs_complete.cpio.gz' into the root of your Linux kernel source directory.

(c) Run '>make ARCH=microblaze menuconfig' in a terminal window as you did last lab. Do not forget to 'source' the 'compile_settings.csh' file and ensure you are in the root directory of the kernel source code when you run menuconfig. Ensure your configuration from last time was loaded.

(d) Within the 'menuconfig' window, select 'General setup'. Scroll down to the bottom of the screen and change the 'Initramfs source files(s)' field to 'initramfs_complete.cpio.gz' (See Figure 1). When the kernel compiles, this setting will cause it to package the kernel with the complete RAM file system rather than the minimalistic one used last time.

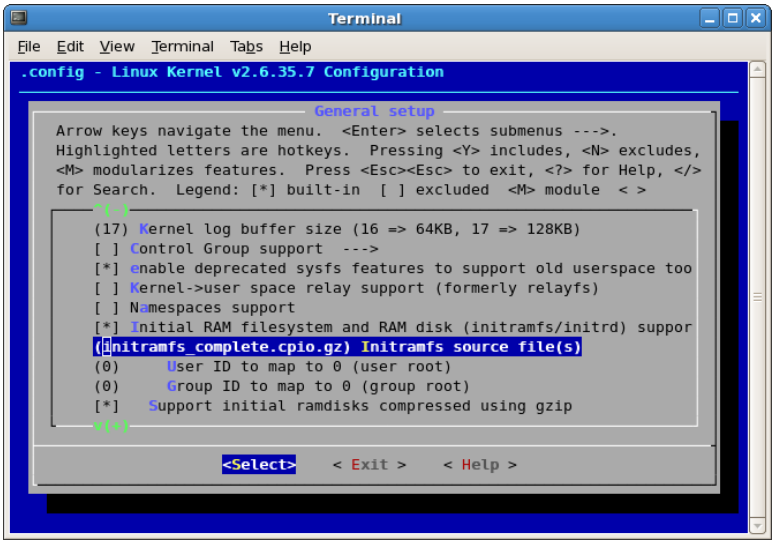

Figure 1: Use Complete RAM file system

(e) Exit out of the 'General setup' screen. From the main menu of 'menuconfig', select 'File systems'. Scroll down towards the bottom of the screen and select 'DOS/FAT/NT Filesystems'. Enable both 'MSDOS fs support' and 'VFAT (Windows-95) fs support' as seen in Figure 2. Note that to properly select an option in 'menuconfig' there must be an asterisk next to the option. When an option is highlighted, you can hit the space bar twice or just press 'y' to select an option.

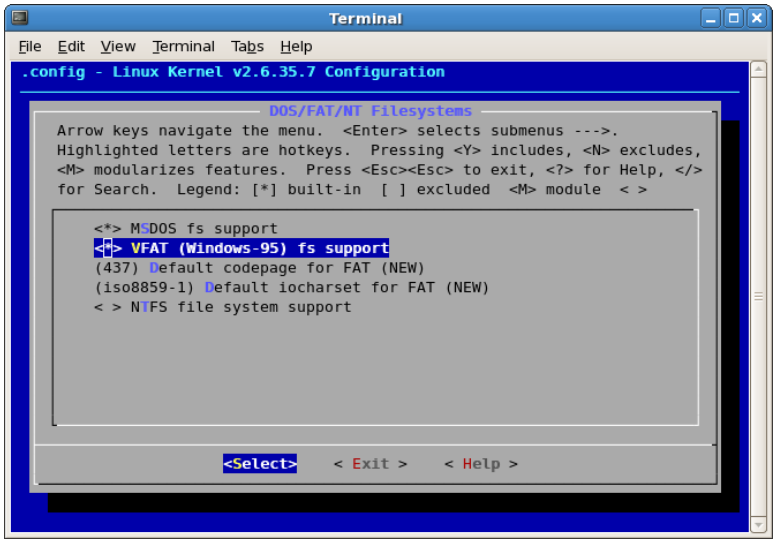

Figure 2: Enable MSDOS and VFAT file system support

(f) Exit out of the current window in 'menuconfig' and select 'Native language support'. Ensure the following options are selected:

- Codepage 437 (United States, Canada)
- ASCII (United States)
- NLS ISO 8859-1 (Latin 1; Western European Languages)
- NLS ISO 8859-15 (Latin 9; Western European Languages with Euro)

These various language support options are required to mount MSDOS and FAT partitions.

(g) Now we must ensure the SysACE block device drivers are enabled in the kernel. This will provide the CF read/write capabilities from within the kernel. Return to the main menu of 'menuconfig' and select 'Device Drivers' and then 'Block devices'. Locate the 'Xilinx SystemACE support' option and ensure it is selected. See Figure 3

(h) Exit out of 'menuconfig' saving the configuration file. Run the following command to compile the kernel with the configuration above:
>make ARCH=microblaze simpleImage.xilinx

(i) Once the kernel is done compiling, create a .ace file as outlined in Lab 4 and boot Linux on the XUP board.
Do not forget to copy the elf containing the recently compiled kernel into your lab5 directory!

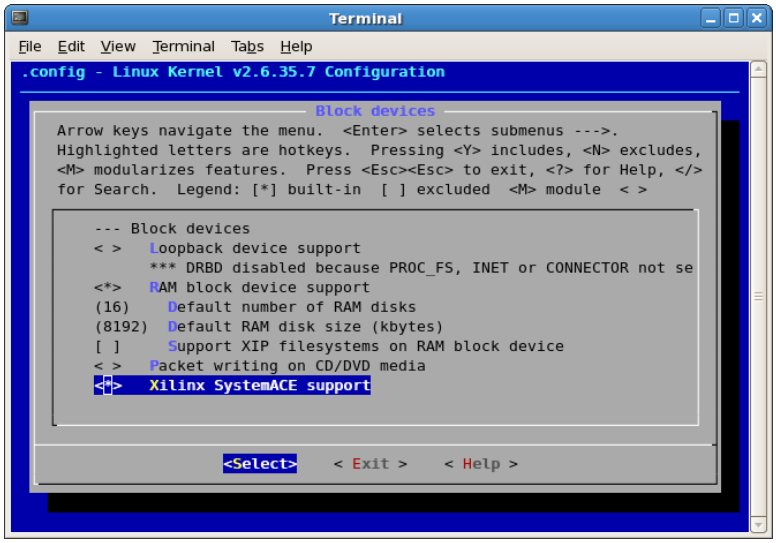

Figure 3: SystemACE Device Driver Support

2. With Linux up and running on the XUP board it is now time to mount the CF card so we can read and write files to it.

   (a) From within the XUP Linux serial console (i.e. kermit on the CentOS machine), run the following commands:
   >mkdir -p /lib/modules/2.6.35.7
   >mount -t vfat /dev/xsa1 /lib/modules/2.6.35.7

   The first command creates a directory called '2.6.35.7' that we will use as a mount point. The '-p' option causes 'mkdir' to create the necessary parent directories. The '/lib' directory does not have a subdirectory called 'modules' so it must be created. The 'modules/2.6.35.7' directory is where our Linux kernel will look for kernel modules to load and unload. Therefore, we will mount the CF card to this location. The last command does just that. The '-t vfat' option tells mount that this is a FAT partition. '/dev/xsa1' is the CF reader device file. '/lib/modules/2.6.35.7' is the mount point we just created.

   (b) Test out the mount operation by running the following commands in kermit:
   >cd /lib/modules/2.6.35.7
   >ls -la

   The first command changes the current directory to the CF mount point. The next command lists the contents of the current directory. Do the files in the 2.6.35.7 directory look familiar?

(c) Now test the write capabilities of the CF card by creating directory within the 2.6.33 directory. Run the following commands in kermit:
>mkdir test
>ls -la
Note the date stamp on the 'test' directory. Demonstrate your progess to the TA.

(d) We can now use the CF card to transfer files between the CentOS workstation where we will compile our kernel modules and the XUP board where we will run our kernel modules. Before removing the CF card from the XUP Linux system, we must un-mount the FAT partition on the CF card to avoid corrupting the file system. Run the following commands in kermit:

>cd /
>umount /lib/modules/2.6.35.7

The first command changes the current directory to the root directory. The purpose of this command is to move out of the mount point prior to un-mounting. The second command un-mounts the FAT partition.

(e) Change the current directory to '/lib/modules/2.6.35.7' using 'cd' and examine the contents of the directory using 'ls'. What do you see?

3. At this point in lab, the XUP board has read/write access to removable non-volatile file system. We will now cross-compile a simple kernel module on the CentOS machine and load it onto the CF card.

(a) Without turning off the power to the XUP board, remove the CF card and insert it into the CentOS machine. Examine the contents of the CF card. Note that the CentOS machine automatically mounts the CF card when inserted.

(b) Create a directory under 'lab5' called 'modules' and copy the text below into a file called 'hello.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 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("Paul V. Gratz (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);
```

(c) Now we need to create a Makefile. In 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.

Note: When compiling the kernel module, information about the original kernel configuration must be know because the kernel module will become an integral part of the kernel when installed. Thus, we must provide the Makefile with a path to the original kernel source.

```
obj-m += hello.o

all:
    make -C <kernel_source_directory> M=$(PWD) modules

clean:
    make -C <kernel_source_directory> M=$(PWD) clean
```

(d) Then type the following command in the terminal window under the 'modules' directory:
>make ARCH=microblaze

(e) If successful, you should see a hello.ko file appear in your 'modules' directory. Copy this file to the CF card.

(f) Properly remove the CF card from the CentOS machine and insert the card into the XUP board. Do not power cycle the the XUP board.

(g) From kermit, execute the '>mount -t vfat /dev/xsa1 /lib/modules/2.6.35.7' command again and change the current directory to the '2.6.35.7' directory.

(h) At this point, we have cross-compiled our Linux kernel module and provided access to it via the XUP board. It is now time to load the module into the Linux kernel on the XUP board. This is done with the following command:
>insmod hello.ko

(i) To see the output of the 'printk' statement that is called when the kernel module is loaded run the following command in the XUP terminal:
>dmesg | tail

Demonstrate this output to the TA.

(j) We can see what modules are loaded by running '>lsmod' in the XUP terminal window. Do this and ensure the 'hello' module is listed.

(k) To remove the module, run '>rmmod hello', and then run '>lsmod' to ensure the module was removed. Also run 'dmesg' again to examine the output of the module during removal.

Deliverables

1. [4 points.] Demo the working kernel module to the TA.

   Submit a lab report with the following items:

2. [5 points.] Correct format including an Introduction, Procedure, Results, and Conclusion.
3. [4 points.] Commented C files.
4. [4 points.] The output of the XUP terminal (kermit) for part 3 of lab.
5. [3 points.] Answers to the following questions:
   (a) If prior to step 3.f, we accidentally reset the XUP board, what additional steps would be needed in step 3.g?
   (b) What is the mount point for the CF card on the CentOS machine?
   Hint: Where does the CF card lie in the directory structure of the CentOS file 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?