ECEN 749: Microprocessor Systems Design

Lab 5: Introduction to Kernel Modules on the Zynq Linux System

Manav Gurumoorthy

830000011

Section 603

Table of Contents

Introduction

Linux's extensibility is due in part to the use of Linux kernel modules that are used to run a variety of applications on different hardware. In this lab, a "Hello World" kernel module was compiled and loaded onto the Zynq board running Linux. Following which another kernel module was written that accessed the multiply IP created in a previous lab, to perform multiplication in hardware and to display the result over a UART console.

Procedure

Configuration

This lab beings from the configuration done in the previous lab. There are no additional steps required once the Linux kernel was compiled for the Zynq board and the board was booted.

Mounting the storage device

In the previous lab, any files written to or created were lost when the board was reset, this was because there was no non-volatile storage connected to the board. In this lab, the kernel module developed on the CentOs machine was to be transferred to the zybo board for execution. For this to be done, the SD card was mounted to the zynq processor so that it could read from and write to the SD card.

The command to mount the board is:

>mount /dev/mmcblk0p1 /mnt/

Once the device was mounted, it could be written to and read from, from the zynq proessor.

Creating the custom Hello World IP:

Now the Zynq processor has read and write access to the non-volatile SD card storage. The next step was to create the Hello World 'C' program. This program demonstrates module initialisation, module release and printk. The module also had access to the multiply IP that was created in the previous lab.

A makefile was then created in order to generate the kernel module from the C file. Once make was run a 'hello.ko' file was created which was copied to the SD card and inserted into the Zybo board. On running the kernel module using

> insmod hello.ko

Running the Kernel Module on the Zynq board:

Once the make command is run on the makefile, it generates a '*.ko' file, the file is put on the SD card. The SD card was then inserted into the Zybo board. On the Zynq processor through the PICOCOM terminal the kernel module was executed. It was executed using

> insmod multiply.ko

Running the command gives the result of the multiply on the PICOCOM console.

Multiply.c code:

```
#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
```

```
    // use ioremap, print the physical and virtual address
      virt_addr = ioremap(PHY_ADDR, MEMSIZE);
      printk("\nThe Physical address is %d", PHY_ADDR);
         printk("\n The Virtual Address is %p",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
    iowrite32(2, virt_addr + 4);
    printk(KERN_INFO "Writing a 2 to register 1\n");
    // use iowrite32

    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);
```

Results

The results of mounting the SD card and running the two kernel modules are shown below.

```
Applications  Places  Terminal

File  Edit  View  Search  Terminal  Help
ehci-pci: EHCI PCI platform driver
zynq-dr e0002000.ps7-usb: Unable to init USB phy, missing?
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
Xilinx Zynq CpuIdle Driver started
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
sdhci-pltfm: SDHCI platform and OF driver helper
sdhci-arasan e0100000.ps7-sdio: No vmmc regulator found
sdhci-arasan e0100000.ps7-sdio: No vqmmc regulator found
mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
ledtrig-cpu: registered to indicate activity on CPUs
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
can: controller area network core (rev 20120528 abi 9)
NET: Registered protocol family 29
can: raw protocol (rev 20120528)
can: broadcast manager protocol (rev 20120528 t)
can: netlink gateway (rev 20130117) max_hops=1
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-a05'
zynq_pm_late_init: Unable to map DDRC IO memory.
Registering SWP/SWPB emulation handler
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
ALSA device list:
  No soundcards found.
RAMDISK: gzip image found at block 0
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
mmc0: card aaaa removed
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SS08G 7.40 GiB
 mmcblk0: p1
Hello Andrew!
random: nonblocking pool is initialized
Goodbye Andrew!
zynq> mmc0: card aaaa removed
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SS08G 7.40 GiB
 mmcblk0: p1

zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
zynq> mount /dev/mmcblk0p1 /mnt/
mount: mounting /dev/mmcblk0p1 on /mnt/ failed: Device or resource busy
zynq> cd /mnt/
zynq> ;s
-/bin/ash: syntax error: unexpected ";"
zynq> ls
BOOT.bin                  test
System Volume Information  uImage
devicetree.dtb            uramdisk.image.gz
hello.ko
zynq> insmod hello.ko
Mapping virtual address...

The Physical address is %
 The Virtual Address is %
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>

Terminal  Home  lab5.pdf - Mozilla Firefox  modules
```

Conclusion

Lab 5 describs the steps to mount an SD card and add kernel modules for execution on the Zynq processing system. We also used the Zybo board to run a user program that interacts with the hardware IP module.

Questions

1. If the board is power cycled in 2.f, the next time we would want to access the SD card file system we would have to run mount again.
2. The SD card was mounted at /run/media/mana/2CCE-FE3C
3. We would have to change the first line in the make file.
   obj -m += newname.o
   Using the kernel directory from lab 4 would not have any consequences as both the linux kernels are compiled for the same hardware.