**Lab #6**

**Objective**: Writing device drivers
**Outcomes:**
After this lab, you will be able to

- Write a simple character driver in memory for ARM Linux
- Write a UART driver for ARM Linux

**Character driver in memory for ARM Linux**

Download the following files from Moodle – memory.c, mem_test.c and Makefile

memory.c is a simple memory based character driver and mem_test.c tests file operations on this driver.

Cross compile mem_test.c into the output file mem_test (see the "to do" in Lab #3). You can compile statically if you wish. The driver is a loadable kernel module as done in Lab #5. Following instructions from Lab #5, make the memory kernel module. The busybox install/home/your_name directory has the files mem_test and the kernel object memory.ko. Now launch Linux under QEMU.

In Linux, devices are accessed from user space in exactly the same way as files are accessed. These device files are normally subdirectories of the /dev directory. To link normal files with a kernel module two numbers are used: major number and minor number. The major number is the one the kernel uses to link a file with its driver. The minor number is for internal use of the device.

To access the device, a file (which will be used to access the device driver) must be created, by typing the following command at the QEMU Linux prompt.

# mknod /dev/memory c 60 0

In the above, c means that a char device is to be created, 60 is the major number and 0 is the minor number.

Within the driver, in order to link it with its corresponding /dev file in kernel space, the register_chrdev function is used. It is called with three arguments: major number, a string of characters showing the module name, and a file_operations structure which links the call with the file functions it defines. Also, note the use of the kmalloc function. This function is used for memory allocation of the buffer in the device driver which resides in kernel space.

memory_read function has the following arguments: a type file structure; a buffer (buf), from which the user space function (fread) will read; a counter with the number of bytes to transfer (count), which has the same value as the usual counter in the user space function (fread); and finally, the position of where to start reading the file (f_pos). In this simple case, the memory_read function transfers a single byte from the driver buffer (memory_buffer) to user space with the function copy_to_user

Similary, memory_write function has the following arguments: a type file structure; buf, a buffer in which the user space function (fwrite) will write; count, a counter with the number of bytes to transfer, which has the same values as the usual counter in the user space function (fwrite); and finally, f_pos, the position of where to start writing in the file. In this case, the function copy_from_user transfers the data from user space to kernel space.

Now load the module,

# insmod memory.ko

and unprotect the device with

# chmod 666 /dev/memory

Execute test to check all the file operations done from user land (see mem_test.c)

# test

This should print the byte value read and written.

**To do:**

Write a Linux device driver for the UART terminals on the emulated QEMU VersatilePB board. This is a simplified character driver which reads and writes to the UART data register by polling; similar to the read/write system call implementation in Lab2(syscalls.c). Test your driver by performing file I/O.

Hints:

1. On ARM all I/O access is memory mapped.

2. Use the function ioremap in the initialization function to assign virtual addresses to the memory regions. See documentation on the VersatilePB board to find out the UART I/O addresses.

3. Use the kernel functions ioread32 and iowrite32 to read and write to these addresses.