ECGR 4101/5101 - Lab 4

Objective: Writing kernel modules

Outcomes:
After this lab, you will be able to

- Write and install a kernel module
- Pass parameters to a kernel module
- Write a kernel module that accesses kernel variable

Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image. Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

Download the Lab 4 zip file from Canvas.
Edit *KERN_SRC* variable in the Makefile to set it to your Linux kernel path from Lab 3.

From Lab4 directory,
```
$ make
```

The kernel module *hello_world_module.ko* is generated. Copy it to the *_install* directory under Busybox from Lab 3.
From the *_install* directory do the following,
```
$ find . | cpio -o --format=newc > ../rootfs.img
$ cd ..
$ gzip -c rootfs.img > rootfs.img.gz
```

From Lab4 directory launch QEMU,

```
$ ../Lab1/qemu-5.1.0/arm-softmmu/qemu-system-arm -M versatilepb -m
128M -kernel ../Lab3/linux-5.8.14/arch/arm/boot/zImage -initrd
../Lab3/busybox-1.25.1/rootfs.img -append "root=/dev/ram rdinit=/bin/sh"
-dtb ../Lab3/linux-5.8.14/arch/arm/boot/dts/versatile-pb.dtb -nographic
```

At the emulated Linux prompt,

```
# mount -t proc proc /proc
# mkdir -p /lib/modules/`uname -r`
# cp hello_world_module.ko /lib/modules/{your kernel version number}
# modprobe hello_world_module  //Loads the module; prints "Hello, world"
# lsmod  //Check if module exists
# modprobe -r hello_world_module  //Removes module; prints "Goodbye, cruel
world"
```

Do the following

- Write a kernel module that accepts as command line parameters the name of the person to be greeted and the number of times the greeting is to be printed.

- The */proc* file system mechanism for the kernel and kernel modules to send information to processes. Originally designed to allow easy access to information about processes (hence the name), it is now used by every bit of the kernel which has something interesting to report, such as */proc/modules* which provides the list of modules and */proc/meminfo* which stats memory usage statistics. Examine the */proc* on your Linux machine (VM or laptop).

- System timers interrupt the processor at programmable frequencies. This frequency, or the number of timer ticks per second, is contained in the kernel variable HZ. The jiffies variable holds the number of times the system timer popped since the system booted. The kernel increments jiffies HZ times every second. Thus, on a kernel with a HZ value of 100, a “jiffy” is a 10-millisecond duration, while on a kernel with HZ set to 1000, a jiffy is only 1-millisecond. Write a kernel module (jiffies module) that uses the jiffies and HZ kernel variables to write the value of time since bootup to */proc*.

See,
https://devarea.com/linux-kernel-development-creating-a-proc-file-and-interfacing-with-user-space/#.X53WxIhKhPY
Also, http://tuxthink.blogspot.com/2013/10/creation-of-proc-entrt-using.html