Linux Kernel Fundamentals: Chapter 3, Working with Loadable Kernel Modules

In this series of challenges, we create a loadable module, experiment with the use of loadable modules, and create and use parameters for modules.

You need to be root.

You need to have installed a Linux kernel development package. On Ubuntu, that is normally linux-headers-$(uname -r). There should be a link to the kernel directory with a Makefile, including subdirectory etc., called /lib/modules/$(uname -r)/build. If the build does not exist or is a broken link, then you don't have everything you need installed.

We are working with Linux kernel code. **Bad things can happen**. It is best to do this with a virtual machine that is OK if it becomes corrupted.

1. Create a loadable module. Make an empty directory to work in.

a. Create a file called lab.c. Add preprocessor commands to include theses two header files: linux/module.h and linux/init.h.
b. Add a function called my_init_module(). This should take no arguments and return an int. This function should use printk() to print a message. my_init_module() should return 0. Register the function with module_init().
c. Create a function called my_cleanup_module() that takes no arguments and has no return value. It should print a message with printk(). Register the function with module_exit().
d. Add a line at the end for the module license, MODULE_LICENSE("GPL");
e. Create a Makefile for making lab.ko.
f. Compile your module to a .ko file by using make.
g. Load your module with insmod. You need to use sudo.
h. What output did you see? *You may need to use the dmesg command.*
i. Run lsmod. Do you see your module?
j. Use rmmod to unload your module. What message did you see with dmesg
k. Run lsmod again. Do you still see your module?

2. Experiment with the return code of init_module().

a. Edit your module and change the return value of init_module() from 0 to -1.
b. Compile the module, and try to reload it with insmod. What error did you get?
c. Does your module show up in the output of lsmod?
d. What happens if you try to unload the module with rmmod?
e. Did my_cleanup_module() ever get called?

3. Experiment with embedded documentation.

a. Modify your module to include the module author and module description.
b. Recompile your module. Run modinfo with the -d and -a options against your module.

4. Add some modifiable parameters.

a. Edit your module. Add both a static int called number and a static char* called word. Initialize number to some integer. Initialize word to some string.
b. Use the module_param() macro to flag number as an integer and word as a string.
c. Edit the module and use the MODULE_PARM_DESC() to give descriptions for both number and word.
d. Recompile your module. Run modinfo -p against the module.
e. Edit the init_module() function. Have it print out the values of number and word with printk().

f. Recompile and load your module. Unload and reload the module while passing new values of number and word as arguments to insmod.