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 CentOS/Red Hat systems, that is normally `kernel-devel`. 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.
    - Create a file called `lab.c`. Add preprocessor commands to include theses two header files: `linux/module.h` and `linux/sched.h`.
    - 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()`.
    - 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()`.
    - Create a Makefile for making `lab.ko`.
    - Compile your module to a `.ko` file by using `make`.
    - Load your module with `insmod`. What output did you see? You may need to use the `dmesg` command or look in `/var/log/messages`.
    - Run `lsmod`. Do you see your module?
    - Use `rmmod` to unload your module. What message did you see?
    - Run `lsmod` again. Do you still see your module?

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

2. Experiment with embedded documentation.
    - Modify your module to include the module author and module description.
    - Recompile your module. Run `modinfo` with the `-d` and `-a` options against your module.

3. Add some modifiable parameters.
    - 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.
    - Use the `module_param()` macro to flag `number` as an integer and `word` as a string.
    - Edit the module and use the `MODULE_PARM_DESC()` to give descriptions for both number and word.
    - Recompile your module. Run `modinfo -p` against the module.
    - Edit the `init_module()` function. Have it print out the values of number and word with `printk()`.
    - Recompile and load your module. Unload and reload the module while passing new values of number and word as arguments to `insmod`.