Operating Systems Homework 3 Report

Andres Ponce, 0616110

2020-12-07

Video Link

TBD

Questions

1. **What is a static kernel module? What is a dynamic kernel module? What is the other name of a dynamic kernel module? What are the differences between system calls and dynamic kernel modules(mention at least 3)?**

A kernel module extends the functionality of the kernel without being part of it. These modules are loaded by the main kernel, so they are not part of the main kernel. We can load and unload these modules by demand by using the `insmod` and `rmmod` commands. Dynamic kernel modules, also known as **loadable kernel modules** can be loaded by the user to add custom functionality to the kernel. Static kernel functions, on the other hand, are loaded at boot time after the intiial ram file system loads the kernel, and are required by the system to boot.

There are some differnces between kernel modules and system calls, which we implemented in our previous project. Firstly, system calls require us to recompile the entire kernel, since they can access some kernel resources. Kernel modules are also located outside the kernels, so modifying, inserting, or removing kernel modules does not require a recompile.

2. **Why does adding a system call require kernel re-compilation, while adding a kernel module does not?**

System calls are part of the kernel. This means that we cannot dynamically load new system kernels during execution of our operating system. System calls would be like adding a new function to a C source code file. In order for the function to be available we have to save the source file and recompile the program.

Kernel modules, on the other hand, are different because they are loaded in from the outside. They don't require the kernel to be recompiled because their source code is not directly inside the kernel. Continuing our C source code example, this would be like having a C program that reads variables from an external file and then adding a variable to that external file. Since we are not changing the source code of the program and instead adding modifications from outside, our program does not require a recompile. 3. **What are the commands `insmod`, `rmmod`, and `modinfo` for? How do you use them?**

These commands all deal with kernel modules. `insmod` `ins`erts a `mod`ule into the Linux kernel. The `rmmod` command removes a module from the Linux kernel, and the `modinfo` command shows the info associated with a particular kernel module.

To use these kenrel modules, we need to know the name of the module, as well as what program options we would like to provide.

4. **Write the usage of the following commands:**

- `module_init`

The `module_init()` system call will load the module into the kernel space. Once it allocates space and can initiaize the module parameters, it will call the module's `initialize()` function, which we pass as an argument. In our module, we print the values of our kernel parameters out to the kernel ring buffer.

- `module_exit`

In contrast to the `module_init()` system call, the `module_exit()` function takes care of wrapping up our module when the module is removed or otherwise terminated. The parameter is a function pointer to the cleanup routine we want to execute. In our case we again reprint the values to the kernel ring buffer, some of which might be changed.

- `MODULE_LICENSE`

The `MODULE_LICENSE()` specifies the license under which our module exists. The "GPL" string represents the "GNU Public License", specifically v2 of that license. This license enables the software to be distributed freely according to the philosophy of the Free Software Foundation. In our project, we use it by placing it at the top of the file.

- `module_param`

The `module_param()` command adds a parameter to our module, and we specify three parameters: the value to be modified, the type, and a mask for the permission. The last value is explained more in detail later.

- `MODULE_PARM_DESC`

This command specifies how to use the module parameter, what it does and how to use it. It is printed out when we inspect the module using `modinfo`, since that will print the descriptions for all the values.

5. **What do the following terminal commands mean? (Explain what they do and what does the `-x` mean in each case):**

- `cat`

The `cat` command prints out the contents of a file to standard output. It can also concatenate files.

- `ls -l`

The `ls` command is one of the most fundamental commands in Linux, and it lists the contents in a directory. When we give it the `-l` option, it will use a "long listing" format, where it indicates the permissions on the directory contents, the user which created them, the group the user belogns to, the size, and the time each was created.

- `dmesg -wH`

The `dmesg` command prints the contents of the kernel ring buffer. This means that any message we write in the `printk()` function will appear in this buffer. The `-wh` options signify two things: that the buffer should wait for new messages (`-w`), and that the contents should be printed out in human readable format (`-H`).

- `lsmod`

The `lsmod` command will list all the currently loaded kernel modules and their status. It formats and prints out the contents of `/proc/modules`.

- `lsmod | grep`

The first command in this sequence is the same as the previous command, however, we "pipe", or send the results of `lsmod` to `grep`. `grep` looks for patterns in standard input, so we can use it to search whether our own kernel modules are running.

6. **There is a -644 in the line**

```
1 module_param(studentId, int, 0644);
```

inside `paramsModule.c`. What does 0644 mean?

7. **What happens if the initialization function of the module returns -1? What type of error do you get?**
8. **In section 1.3 - step 6, `modinfo` hsows the information of some variables inside the module but two of them are not displayed. Why is it?**
9. **What is the `/sys/module` folder for?**
10. **In section 1.3 (`paramsModule.c`), the variable `charparameter` is of type `charp`. What is `charp`?**