University of Colorado at Colorado Springs
Operating Systems Project 2
Kernel Modules and Processes

Instructor: Serena Sullivan
Total Points: 100
Out: 9/12/2022
Due: 11:59 pm, 9/27/2022

Introduction

The purpose of this project is to study how to write Linux kernel modules and learn how processes are managed by the Linux OS. The objectives of this project is to learn:

1. How to write a helloworld Linux kernel module. The free book The Linux Kernel Module Programming Guide will be super helpful.
2. How to obtain various information for a running process from the Linux kernel.

Project submission

For each project, please create a zipped file containing the following items, and submit it to Canvas.

1. A report that includes (1) the (printed) full names of the project members, and the statement: **We have neither given nor received unauthorized assistance on this work**; (2) the name of your virtual machine (VM), and the password for the `root` user[1] (`root` is required to test kernel modules); and (3) a brief description about how you solved the problems and what you learned. The report can be in **txt, doc, or pdf format**. A Word template will be provided.
2. Your code and `Makefile`; **please do not include compiled output**. Even though you have your code in your VM, submitting code in Canvas will provide a backup if we have issues accessing your VM.

When you are ready, turn over to the next page.

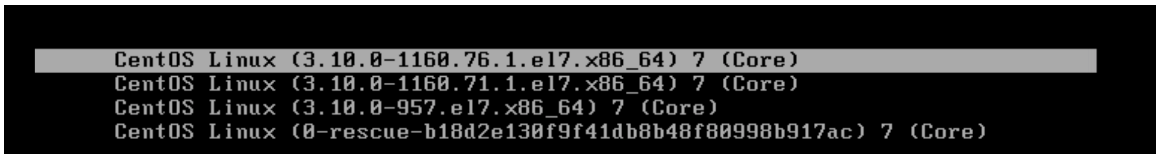
[1]If at any time you forget your root password, there's a solution (that I don't recommend for security reasons).

Project

Part 0.0: Use the provided template for report (5 points)

Part 0.1: Preparation

Like in Project 1, please boot the newest version of the kernel by choosing it at boot time, as shown below.



```
CentOS Linux (3.10.0-1160.76.1.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-1160.71.1.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-957.el7.x86_64) 7 (Core)
CentOS Linux (0-rescue-b18d2e130f9f41db8b48f80998b917ac) 7 (Core)
```

Next, run the following command as **root** to install Linux headers of your particular kernel version so you can implement kernel modules successfully:

```
# yum install -y kernel-devel kernel-headers
```

Due to historic reasons, specific versions of headers may not be present in the repositories, which is the case of this CentOS distribution's old kernel (3.10.0-957.el7.x86_64). See this thread. But the headers for the new version are available. Therefore 🧸 with me and boot the new kernel every time when working on kernel modules.

Part 1: Create a helloworld kernel module (25 points)

The following code is a complete helloworld module. Be careful about the quotation marks – type the `printk` function by hand rather than copy&pasta to avoid encoding issues. Save the code as `hello.c`.

```c
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void) {
    printk(KERN_INFO "Hello world!\n");
    return 0;
}

void cleanup_module(void){
    printk(KERN_INFO "Goodbye world!\n");
}

module_init(init_module);
module_exit(cleanup_module);

MODULE_LICENSE("GPL");
```

Note that in the last line `MODULE_LICENSE("GPL")`, there's an underscore between `MODULE` and `LICENSE`. This module defines two functions. `init_module` is invoked when the module is loaded into the kernel and `cleanup_module` is called when the module is removed from the kernel. `module_init` and `module_exit` are special kernel macros to indicate the role of these two functions.

Use the following `Makefile` to compile the module (you need **root** permission).

```
obj-m += hello.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Do you see any errors when you `make`? If so, how to fix the error (5pts)? After fixing, you will need to load your module into the kernel. One of the files generated by `make` is `hello.ko`. To load the module, run this command as **root**:

```
# insmod hello.ko
```

Then use this to see the output:

```
# dmesg -T | tail
```

Use the following command to verify the module has been loaded:

```
# lsmod
```

To remove the module from the kernel do the following (notice `.ko` is not needed):

```
# rmmod hello
```

Use `dmesg` again to see the module's output:

```
# dmesg -T | tail
```

`dmesg` displays the kernel ring buffer, from almost everything (more info about kernel ring buffer). You need to look at the most recent event logs to see messages from your module. The `-T` option will print the timestamp as human readable time; piping the output to `tail` will print the last 10 lines. If you want to see the last 20 lines, provide an option to `tail` as `tail -20`. Notice the different timestamps when `Hello world!` and `Goodbye world!` are printed (15pts).

If I want to print the messages from `init_module` and `cleanup_module` to the standard output in addition to the kernel ring buffer, what should I change (5pts)?

Part 2: Create a print_self kernel module (30 points)

Follow the instructions above and implement a module `print_self`. This module identifies the current process at the user-level and print out various information of the process. Implement the `print_self` module and print out the following:

- Process name, id, and state;
- The same above information of its parent processes until `init`.

To keep things clean, I recommend that you create different sub-directories for different modules and create a different `Makefile` than Part 1. In your report, please (1) list steps to load and remove your module, and read your module's output; and (2) answer the following questions:

1. The macro `current` returns a pointer to the `task_struct` of the current running process. See the following link: `https://linuxgazette.net/133/saha.html` When you load your module, which process is recognized as `current`?
2. As discussed in our lecture, in old kernels the mother of all processes is called `init`. In newer kernels, what is it called and what do you see from your module's output?
3. To see the different states of a process, please refer to the same page above `https://linuxgazette.net/133/saha.html` When printing state in your code, please map

the numeric state to its string state, e.g., print `TASK_RUNNING` if state is 0. From your module's output, which state(s) are observed?

Part 3: Create a print_other kernel module (30 points)

Implement another module `print_other` to print the information for an arbitrary process. The module takes a process PID as its argument and outputs the above information (same as Part 2) of this process all the way back to `init`.

Some hints:

1. There are some functions and macros that the Linux kernel provides to look up a process in the process table. Please refer to the lecture slides. However, not all of the functions and macros are available to modules. See this very old thread as an example.
2. To get a valid PID for a process that's active, you may use command `pgrep bash`. `bash` is the default command line shell. `pgrep bash` will provide PIDs of `bash`. You can choose any of them if you get more than one.
3. Refer to the free book The Linux Kernel Module Programming Guide how to pass an argument to your kernel module. In your report, please list steps to load and remove your module, and read your module's output.

Part 4: Kernel Modules and System Calls (10 points)

So what exactly is a kernel module? Please read this short article (kernel modules are also known as loadable modules). Use your own words, please answer the following:

1. What's the difference between a kernel module and a system call?
2. This article is over 20 years old. If you try this example from the article in your VM, does it still work? Use your own words to explain why you think this may be a good (or bad) thing.