CoE 135 Lab 6 1stSY1920

Kernel Modules and Kernel Data Structures

Objectives:

- Learn how to create a simple Linux Kernel Module
- Learn how to insert and remove a Linux Kernel Module
- Navigate Linux Source Code
- Understand the fundamental differences between User space programs and Kernel space programs
- Learn how to make use of Linux Kernel Data Structures
- Introduce the Linux Kernel Data Structure that represents processes

General Specifications:

Install the linux kernel headers specific to your current Linux installation using the following command:

```
sudo apt-get install linux-headers-$(uname –r)
```

Use the template uploaded with this document. The first file is *helloworld.c* which creates a kernel module that prints “Hello World!” in *dmesg*. *Makefile* is used to compile kernel modules. Change *obj-m* to proper file to compile and the *KDIR* should point to your kernel modules directory.

Steps to run your kernel module in terminal:

1. `make` (in directory where Makefile is stored)
2. `sudo insmod helloworld.ko` (insert module to run your module)
3. `sudo rmmod helloworld.ko` (remove module to terminate your module)
4. `make clean` (to clean up Makefiles created)

Part I. Introduction to Kernel Modules

Your first task in this Machine Exercise is to create a simple Loadable Kernel Module that displays the content of the first 5 lines of the /proc/cpuinfo file without reading this file directly. To accomplish this, you will need to search Linux Kernel code to find where these information are stored. A sample output is presented below. You should see this output when you invoke the command dmesg. Save your code as lab6_cpuinfo_lastname.c (e.g. lab6_cpuinfo_briones.c).

Sample Output:

```
processor : 0 vendor_id : GenuineIntel
cpu family : 6
model : 69
model name : Intel(R) Core(TM) i5-4278U CPU @ 2.60GHz
```

Part II. Background and Concept of Data Structures

We have discussed in the lecture that the Linux Kernel makes use of robust data structures such as stacks and queues to model various information used by our Operating System. In this part of the Machine Exercise, we explore the circular, doubly linked list that is available to kernel developers. We will make use of the <*linux/list.h*> library for the list data structure.

Initially, you must define a struct containing the elements that are to be inserted in the linked list. The following C struct defines birthdays:

```
struct birthday {
     int day;
     int month;
     int year;
     struct list_head list;
}
```

Notice the member struct list_head list. The list_head structure is defined in the include file <*linux/types.h*>. Its intention is to embed the linked list within the nodes that comprise the list. This list_head structure is quite simple, it merely holds two members, next and prev, that point to the next and previous entries in the list. By embedding the linked list within the structure, Linux makes it possible to manage the data structure with a series of macro functions. Save your code as lab6_linkedlist_lastname.c (e.g. lab6_linkedlist_briones.c). Check out Chapter 2 of our main reference book to see how to insert, traverse, and delete items from this kernel linked list data structure. It is somewhere around the end of the chapter.

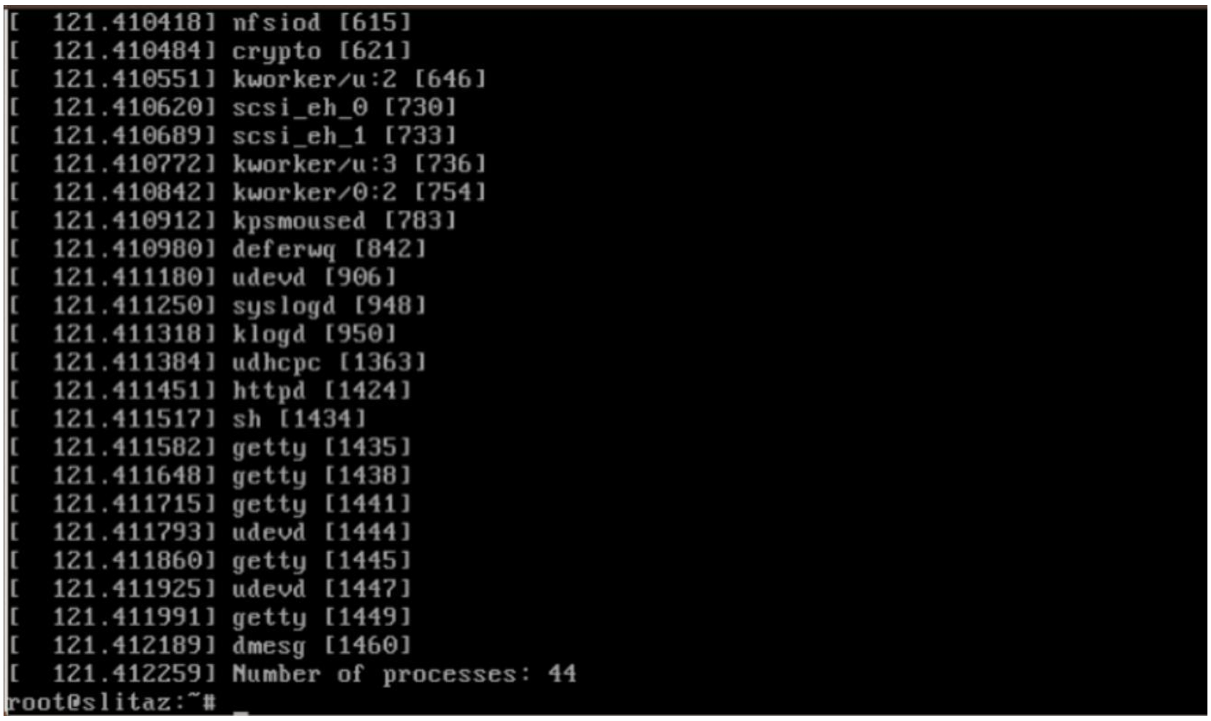
Part III. Kernel Data Structures I

In the module entry point (the initialization function), create a linked list containing five `struct_birthday` elements. Traverse the linked list and output its contents to the kernel log buffer. Invoke the `dmesg` command to ensure the list is properly constructed once the kernel module has been loaded.

In the module exit point (the exit function), delete the elements from the linked list and return the free memory back to the kernel. Again, invoke the `dmesg` command to check that the list has been removed once the kernel module has been unloaded. Save your code as lab6_linkedlist_lastname.c (e.g. lab6_linkedlist_briones.c) – same file as part 2.

Part IV. Kernel Data Structures II

In this part of the exercise, your goal is to write a kernel module that lists all current tasks in a Linux system. You will make use of <*linux/sched.h*> and the `struct task_struct` data structure in this exercise. The output of this program is like the output of the "ps -el" terminal command but is not necessarily the same. This task is quite like Part II. Save your code as lab6_process_lastname.c (e.g. lab6_process_briones.c). Sample output is presented below:



```
[   121.410418] nfsiod [615]
[   121.410484] crypto [621]
[   121.410551] kworker/u:2 [646]
[   121.410620] scsi_eh_0 [730]
[   121.410689] scsi_eh_1 [733]
[   121.410772] kworker/u:3 [736]
[   121.410842] kworker/0:2 [754]
[   121.410912] kpsmoused [783]
[   121.410980] deferwq [842]
[   121.411180] udevd [906]
[   121.411250] syslogd [948]
[   121.411318] klogd [950]
[   121.411384] udhcpc [1363]
[   121.411451] httpd [1424]
[   121.411517] sh [1434]
[   121.411582] getty [1435]
[   121.411648] getty [1438]
[   121.411715] getty [1441]
[   121.411793] udevd [1444]
[   121.411860] getty [1445]
[   121.411925] udevd [1447]
[   121.411991] getty [1449]
[   121.412189] dmesg [1460]
[   121.412259] Number of processes: 44
root@slitaz:~# _
```

Deadline: Week of 30 September 2019

- Grades are given DURING the laboratory class hours. Students who failed to let the lecturer/student assistant check their work during class hours will be given a grade of 0.
- Codes are expected to run properly and will be checked on Ubuntu 18.04 LTS. You may also use your own Arch Linux installed from Lab 1 exercise for checking, given that you bring it on your checking day.
- Name your files as lab6_module_lastname (e.g. lab6_cpuinfo_briones). Also, put your name at the start of your code. Incorrect name syntax will be given a deduction of 10%.
- Submit your files to Google Classroom.
- Compile your codes with –W –Wall. Codes with warnings (unused parameters, incompatible data types, etc) will be given a deduction of 10%.
- I will be running your c codes through a code checker so make sure you did not copy your code from the internet or from other students.
- Students who are caught cheating will be given a grade of 0 and will be filed a case in the Student Disciplinary Council.

Grading system:

cpuinfo module 30%
linked list module 35%
process module 35%