CSE 422S: Studio 5

Loadable Kernel Modules

Xingjian Xuanyuan

March 11, 2022

In this studio, we will:

1. *Build and install kernel modules.*
2. *Write a kernel module and use it to observe changes in a kernel variable.*

```c
/* simple_module.c - a simple template for a loadable kernel module in
 * Linux, based on the hello world kernel module example on pages 338-339
 * of Robert Love's "Linux Kernel Development, Third Edition."
 */

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

/* init function - logs that initialization happened, returns success */
static int
simple_init(void)
{
    printk(KERN_ALERT "simple module initialized\n");
    return 0;
}

/* exit function - logs that the module is being removed */
static void
simple_exit(void)
{
    printk(KERN_ALERT "simple module is being unloaded\n");
}

module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("LKD Chapter 17");
MODULE_DESCRIPTION ("Simple CSE 422S Module Template");
```

Compiling the above program file involves the following steps:

1. Assuming you are using a PC laptop/desktop, on your own machine create a directory to hold your own kernel modules. In that directory also create a `Makefile` that contains the line: `obj-m := simple_module.o`.

2. Build the module by issuing the commands[1]:

```
module add raspberry
LINUX_SOURCE=<path to your Linux kernel source code>
```

3. Finally, compile via

```
make -C $LINUX_SOURCE ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
↪  M=$PWD modules
```

Successful compilation should produce a kernel module file named `simple_module.ko`.

Next, boot up your Raspberry Pi, open up a terminal window, create a directory to hold your own kernel modules, and use `sftp` to get the `simple_module.ko` file. The `kmod` program provides user-space utilities for managing Linux kernel modules. You will be using three commands in your Pi's terminal:

Usage	Description
`insmod [filename] [module options]`	Insert a module into the Linux kernel
`rmmod [-f] [-s] [-v] [modulename]`	Remove a module from the Linux kernel
`lsmod`	Show the status of modules in the Linux kernel

A module runs in kernel space while an application runs in user space. The terms of kernel space and user space "encompass not only the different privilege levels inherent in the two modes, but also the fact that each mode can have its own memory mapping—its own address space—as well." (Corbet et al. 2005, p. 20) Therefore, the above three commands should be used with `sudo`.

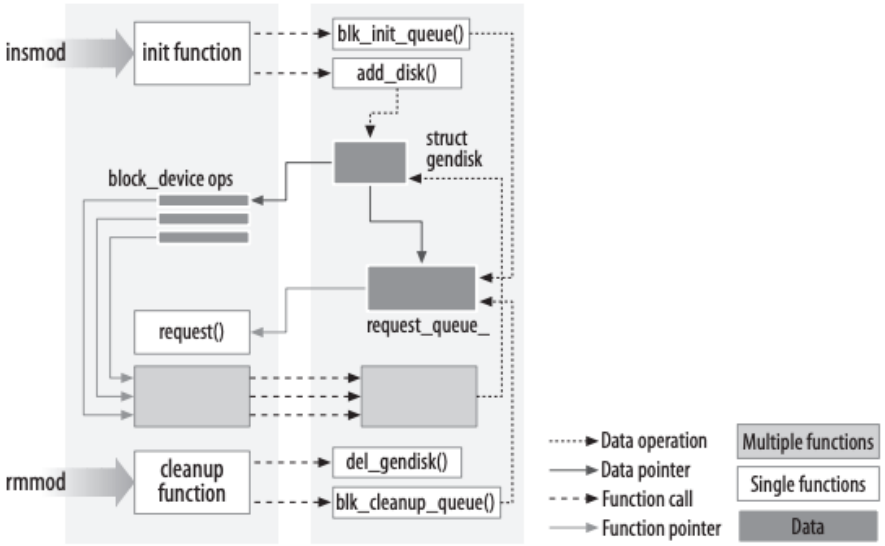

Figure 1: Linking a module to the kernel (Corbet et al. 2005, p. 19)

To see the interval between the time your module was loaded and the time it was unloaded, you can make use of the `jiffies` variable. Create a new file called `jiffies_module.c` to generate system log messages that give the values of `jiffies`:

```
1 #include <linux/init.h>
2 #include <linux/module.h>
3 #include <linux/kernel.h>
4 #include <linux/jiffies.h>
5
```

[1] In my case, the Linux kernel source code is located at `/linux_source/linux`.

```
6  unsigned long module_start, module_end;
7
8  static int
9  jiffies_init(void)
10 {
11     module_start = jiffies;
12     printk(KERN_INFO "jiffies module initialized\njiffies = %lu\n",
       ↪  module_start)
13     return 0;
14 }
15
16 static void
17 jiffies_exit(void)
18 {
19     module_end = jiffies;
20     printk(KERN_INFO "jiffies module unloaded\njiffies = %lu\n",
       ↪  module_end)
21 }
22
23 module_init(jiffies_init);
24 module_exit(jiffies_exit);
25
26 MODULE_LICENSE ("GPL");
27 MODULE_AUTHOR ("LKD Chapter 17");
28 MODULE_DESCRIPTION ("Simple CSE 422S Module Template");
```

References

Corbet, J., Rubini, A. & Kroah-Hartman, G. (2005), *Linux Device Drivers*, 3 edn, O'Reilly Media.

Love, R. (2010), *Linux Kernel Development*, 3 edn, Addison-Wesley.