Introduction to Embedded Software

Practice #3

Linux kernel modules

Dongkun Shin
Embedded Software Laboratory
Sungkyunkwan University
http://nyx.skku.ac.kr/

Objective

- **Compile and test** standalone **kernel modules**, which code is outside of the main Linux sources.
- **Write a kernel module** with several capabilities, including module **parameters**.
- **Access kernel internals** from your module.
- **Create a kernel patch.**

Day Practice

1) **dmesg Screenshot**
   - **Load module (insmod)**
     ✓ Your name , Your student ID (parameter)
   - **Unload module (rmmod)**
     ✓ Time information
       - how many seconds elapsed since you loaded it
2) **Kernel patch**
   - Screenshot or file

Day Practice: submit to i-campus (- 04/07 23:59)

What is kernel module?

What is a kernel module?

- Modules are pieces of code that can be loaded and unloaded into the kernel upon **demand**.
- They extend the functionality of the kernel without the need to reboot the system.

- Without modules,
  - Disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

lsmod

- You can see what modules are already loaded into the kernel by running ***lsmod***
  - ***lsmod*** gets its information by reading the file /proc/modules

```
root@eslab-System-Product-Name:/home/eslab# lsmod
Module                  Size  Used by
nbd                    32768  3
xt_CHECKSUM            16384  1
iptable_mangle         16384  1
ipt_MASQUERADE         16384  3
nf_nat_masquerade_ipv4    16384  1 ipt_MASQUERADE
iptable_nat            16384  1
nf_nat_ipv4            16384  1 iptable_nat
nf_nat                 28672  2 nf_nat_masquerade_ipv4,nf_nat_ipv4
nf_conntrack_ipv4      16384  5
nf_defrag_ipv4         16384  1 nf_conntrack_ipv4
xt_conntrack           16384  1
nf_conntrack          131072  6 nf_conntrack_ipv4,ipt_MASQUERADE,nf_nat_masquerade_ipv4,xt_cor
ntrack,nf_nat_ipv4,nf_nat
libcrc32c              16384  2 nf_conntrack,nf_nat
ipt_REJECT             16384  2
nf_reject_ipv4         16384  1 ipt_REJECT
xt_tcpudp              16384  6
bridge                143360  0
stp                    16384  1 bridge
llc                    16384  2 bridge,stp
ebtable_filter         16384  0
ebtables               32768  1 ebtable_filter
ip6table_filter        16384  0
ip6_tables             28672  1 ip6table_filter
iptable_filter         16384  1
ip_tables              24576  3 iptable_mangle,iptable_filter,iptable_nat
x_tables               40960  11 ipt_REJECT,iptable_mangle,ip_tables,ebtables,iptable_filter,x
```

Writing module

Write hello module

```
$ cd linux
$ mkdir test
$ vi hello_version.c
```

Kernel build must be done first.

```c
/*
* hello_version.c - write module
*/

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

static int __init hello_init(void)
{
    printk("hello world! [YOUR NAME]\n");
    return 0;
}

static int __exit hello_exit(void)
{
    printk("Goodbye world! [YOUR NAME]\n");
    return 0;
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module");
MODULE_AUTHOR("YOUR NAME");
```

Replace to YOUR NAME for day practice

Building your module

$ vi Makefile

```
obj-m := hello_version.o
KDIR := /home/eslab/saftl/simul/new_part/ESW/linux      # Your kernel path

default:
    make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C $(KDIR) M=$(shell pwd) modules

clean:
    make -C $(KDIR) M=$(PWD) clean
```

$ make

Output: hello_version.ko

Testing your module

$ scp hello_version.ko pi@115.145.208.000 :~/ **# your rasp ip address**

```
pi@raspberrypi:~ $ ls
hello_version.ko
```

$ ssh pi@115.145.208.000 :~/ # or use putty
$ sudo insmod hello_version.ko
$ lsmod | grep hello_version

```
pi@raspberrypi:~ $ lsmod | grep hello
pi@raspberrypi:~ $ sudo insmod hello_version.ko
pi@raspberrypi:~ $ lsmod | grep hello
hello_version          16384  0
```

$ sudo rmmod hello_version
$ dmesg | tail –n 5

```
pi@raspberrypi:~ $ sudo rmmod hello_version
pi@raspberrypi:~ $ dmesg | tail -n5
[    9.802630] Bluetooth: BNEP socket layer initialized
[  209.222159] usb 1-1.3: USB disconnect, device number 4
[ 5440.105369] hello_version: loading out-of-tree module taints kernel
[ 5440.106493] hello world! YJK
[ 5804.931751] Goodbye world! YJK
```

Adding a parameter to your module

$ vi hello_param.c

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

/* A couple of parameters that can be passed in: how
times we say hello world  and your student id */

/*
* module_param(foo, int, 0000)
* The first param is the parameters name
* The second param is it's data type
* The final argument is the permissions bits,
*/
static char *id = "yourid";
module_param(id, charp, 0);
static int howmany = 1;
module_param(howmany, int, 0);
```

```c
static int __init hello_init(void)
{
    int i;
    for (i = 0; i < howmany; i++)
        printk("hello world! [YOUR NAME] %s\n", id);

    return 0;
}

static int __exit hello_exit(void)
{
    printk("Goodbye world! [YOUR NAME] %s\n", id);
    return 0;
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module");
MODULE_AUTHOR("YOUR NAME");
```

$ vi Makefile

$ make

Output: hello_param.ko

```
obj-m := hello_version.o hello_param.o
```

Adding a parameter to your module

```
$ sudo insmod hello_param.ko id=“2009123456” howmany=3
$ dmesg | tail –n 10
```

```
pi@raspberrypi:~ $ sudo insmod hello_param.ko id="2009123456" howmany=3
pi@raspberrypi:~ $ dmesg | tail -n10
[    9.802602] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    9.802612] Bluetooth: BNEP filters: protocol multicast
[    9.802630] Bluetooth: BNEP socket layer initialized
[  209.222159] usb 1-1.3: USB disconnect, device number 4
[ 5440.105369] hello_version: loading out-of-tree module taints kernel.
[ 5440.106493] hello world! YJK
[ 5804.931751] Goodbye world! YJK
[ 6559.509781] hello world! [YOUR NAME] 2009123456
[ 6559.509792] hello world! [YOUR NAME] 2009123456
[ 6559.509797] hello world! [YOUR NAME] 2009123456
```

Adding time information

- Improve your module, so that when you **unload** it,
  it tells you **how many seconds elapsed since you loaded it.**

- Can use the **do_gettimeofday()** function

< Day Practice 1 >

dmesg screenshot

- insmod: your name, your student ID (parameter)
- rmmod: how many seconds elapsed since you loaded it

Create a kernel patch

- **Git config**

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Create a kernel patch

- **Create new git branch**

```
$ git checkout –b hello        # your pwd: linux/
```

- **Commit your code**

```
$ git add –A                   # -A means all modified files
$ git commit –m “hello module”
$ git log
```

```
commit d8c0add4e275d13d11a886be8428a08c6f61cf77
Author: Yunji Kang <oso0931@gmail.com>
Date:   Tue Apr 2 14:03:41 2019 +0900

    hello module
```

Create a kernel patch

- **Create patch file**

```
$ git format-patch rpi-4.14.y
```

Output: 0001-hello-module.patch

```
From d8c0add4e275d13d11a8        a08c6f61cf77 Mon Sep 17 00:00:00 2001
From: Yunji Kang
Date: Tue, 2 Apr 2019 14:03:41 +0900
Subject: [PATCH] hello module

---
 test/Makefile        |  8 ++++++++
 test/hello_param.c   | 43 ++++++++++++++++++++++++++++++++++++++++++++++++
 test/hello_version.c | 22 ++++++++++++++++++++++
 3 files changed, 73 insertions(+)
 create mode 100644 test/Makefile
 create mode 100644 test/hello_param.c
 create mode 100644 test/hello_version.c
```

< Day Practice 2 >

Submit your patch file

Next class

- Please bring for next class
  - wii nunchuck
  - Wii WiiChuck Nunchuck adapter