МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

КАТЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Архітектура комп’ютерів

Лабораторна робота №5

«Розроблення модулів Linux Kernel (частина 2)»

Виконав:
студент групи ІО-32
Крадожон М.

Перевірив(-ла):
Каплунов А.

2023

Лабораторна робота №5

Мета

Ознайомитися з механізмом розділення функціоналу між двома модулями ядра Linux, навчитися експортувати функції між модулями, працювати з параметрами модулів і вимірювати час виконання коду у ядрі.

Хід роботи

Підготовка середовища

Для виконання роботи використано контейнер Docker із QEMU (створений у лабораторній роботі №3).
У контейнері вже зібрано ядро ARM і є файлову систему rootfs.cpio.gz.

Перевірка каталогу ядра:

```
ls /workspace/repos/linux-stable
```

Створення модуля ядра

Проект розділено на два модулі:

1. **hello1.ko**
   - Виконує всю роботу зі списком структур, що зберігають час виклику функції.
   - Додано два поля типу `ktime_t` для вимірювання часу до і після друку.
   - Експортує функцію `print_hello()`, яку використовує hello2.ko.
   - Використано `kmalloc` для виділення пам’яті та `kfree` для її звільнення.
2. **hello2.ko**
   - Приймає параметр `times` (кількість викликів функції друку).
   - Використовує функцію `print_hello()` з hello1 для виведення повідомлень.
   - Обробляє параметри:
     - За замовчуванням `times = 1`
     - Якщо `times = 0` або між 5 і 10 → `pr_warn` і робота продовжується
     - Якщо `times > 10` → `pr_err` і модуль не завантажується

Заголовковий файл `hello1.h` винесено у підкаталог `inc` і підключено через `ccflags-y` у Makefile, щоб файли *.c могли використовувати директиву:

Файл hello1.h

```
#ifndef HELLO1_H
#define HELLO1_H

#include <linux/ktime.h>

void print_hello(void);

struct hello_data {
    struct list_head list;
    ktime_t start_time;
    ktime_t end_time;
};

#endif
```

Файл hello1.c

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/list.h>
#include <linux/slab.h>
#include <linux/ktime.h>
#include "hello1.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Maxim Kradozhon");
MODULE_DESCRIPTION("Hello1 module with time measurement");

static LIST_HEAD(hello_list);

void print_hello(void)
{
    struct hello_data *data = kmalloc(sizeof(*data), GFP_KERNEL);
    if (!data)
        return;

    data->start_time = ktime_get();
    pr_info("Hello, world!\n");
    data->end_time = ktime_get();

    list_add_tail(&data->list, &hello_list);
}
EXPORT_SYMBOL(print_hello);

static void __exit hello1_exit(void)
{
    struct hello_data *data, *tmp;

    list_for_each_entry_safe(data, tmp, &hello_list, list) {
        s64 delta = ktime_to_ns(ktime_sub(data->end_time, data->start_time));
        pr_info("Time for print_hello(): %lld ns\n", delta);
        list_del(&data->list);
        kfree(data);
    }

    pr_info("hello1 unloaded\n");
}

module_exit(hello1_exit);
```

Файл hello2.c

```c
#include <linux/init.h>
#include <linux/module.h>
#include "hello1.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Maxim Kradozhon");
MODULE_DESCRIPTION("Hello2 module that calls hello1’s function");

static uint count = 1;
module_param(count, uint, 0660);
MODULE_PARM_DESC(count, "Number of times to print 'Hello, world!'");

static int __init hello2_init(void)
{
    int i;

    if (count == 0 || (count >= 5 && count <= 10))
        pr_warn("Warning: count=%u is unusual, continuing...\n", count);
    else if (count > 10) {
        pr_err("Error: count too large (%u)\n", count);
        return -EINVAL;
    }

    for (i = 0; i < count; i++)
        print_hello();

    pr_info("hello2 loaded with count=%u\n", count);
    return 0;
}

static void __exit hello2_exit(void)
{
    pr_info("hello2 unloaded\n");
}

module_init(hello2_init);
module_exit(hello2_exit);
```

Makefile

```
obj-m += hello1.o
obj-m += hello2.o

ccflags-y := -I$(PWD)/inc

all:
	make -C /workspace/repos/linux-stable M=$(PWD) modules

clean:
	make -C /workspace/repos/linux-stable M=$(PWD) clean
```

Компіляція модуля

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

Результат:

```
root@ebe046d6188a:/workspace/hello# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
make -C /workspace/repos/linux-stable M=/workspace/hello modules
make[1]: Entering directory '/workspace/repos/linux-stable'
  CC [M]  /workspace/hello/hello1.o
  CC [M]  /workspace/hello/hello2.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /workspace/hello/hello1.mod.o
  LD [M]  /workspace/hello/hello1.ko
  CC      /workspace/hello/hello2.mod.o
  LD [M]  /workspace/hello/hello2.ko
make[1]: Leaving directory '/workspace/repos/linux-stable'
```

Додавання модуля до rootfs

```
mkdir rootfs
cd rootfs
gunzip -c ../repos/busybox/rootfs.cpio.gz | cpio -idmv
cp ../hello/hello1.ko ../hello/hello2.ko ../hello/inc/hello1.h ./root
find . | cpio -o -H newc | gzip > ../rootfs_new.cpio.gz
cd ..
rm -rf rootfs
cp ./rootfs_new.cpio.gz ./repos/busybox/
```

Запуск QEMU

```
cd /workspace/repos/busybox

qemu-system-arm -kernel _install/boot/zImage -initrd rootfs_new.cpio.gz \
-machine virt -nographic -m 512 \
--append "root=/dev/ram0 rw console=ttyAMA0,115200 mem=512M"
```

Перевірка роботи модуля

Усередині QEMU:

```
/ # cd root/
~ # insmod hello1.ko
[   48.068161] hello1: loading out-of-tree module taints kernel.
~ # lsmod
hello1 16384 0 - Live 0xbf000000 (O)
~ # insmod hello2.ko
[   64.815025] Hello, world!
[   64.815345] hello2 loaded with count=1
~ # lsmod
hello2 16384 0 - Live 0xbf007000 (O)
hello1 16384 1 hello2, Live 0xbf000000 (O)
~ # rmmod hello2
[   89.875131] hello2 unloaded
~ # insmod hello2.ko count=0
[  207.689518] Warning: count=0 is unusual, continuing...
[  207.690033] hello2 loaded with count=0
~ # rmmod hello2
[  310.584410] hello2 unloaded
~ # insmod hello2.ko count=6
[  312.886434] Warning: count=6 is unusual, continuing...
[  312.886935] Hello, world!
[  312.887191] Hello, world!
[  312.887440] Hello, world!
[  312.887663] Hello, world!
[  312.887865] Hello, world!
[  312.888070] Hello, world!
[  312.888364] hello2 loaded with count=6
~ # rmmod hello2
[  346.796781] hello2 unloaded
~ # insmod hello2.ko count=11
[  352.126742] Error: count too large (11)
insmod: cant insert 'hello2.ko': invalid parameter
~ # rmmod hello2
rmmod: remove 'hello2': No such file or directory
~ # rmmod hello1
[  410.729232] Time for print_hello(): 272752 ns
[  410.729720] Time for print_hello(): 258736 ns
[  410.730119] Time for print_hello(): 239888 ns
[  410.730570] Time for print_hello(): 239472 ns
[  410.731125] Time for print_hello(): 219600 ns
[  410.731611] Time for print_hello(): 196944 ns
[  410.732069] Time for print_hello(): 201792 ns
[  410.732481] Time for print_hello(): 188608 ns
[  410.733557] hello1 unloaded
[  463.986026] hello2: Unknown symbol print_hello (err -2)
insmod: cant insert 'hello2.ko': unknown symbol in module or invalid parameter
```

```
~ # modinfo hello2.ko
filename:       hello2.ko
author:         Maxim Kradozhon
description:    Hello2 module that calls hello1's function
license:        GPL
parm:           count:Number of times to print 'Hello, world!'
depends:        hello1
vermagic:       4.19.325 SMP mod_unload ARMv7 p2v8
~ # modinfo hello1.ko
filename:       hello1.ko
author:         Maxim Kradozhon
description:    Hello1 module with time measurement
license:        GPL
depends:
vermagic:       4.19.325 SMP mod_unload ARMv7 p2v8
~ # cat /sys/module/hello2/parameters/count
3
~ # insmod hello1.ko
~ # insmod hello2.ko count=3
[  610.968931] Hello, world!
[  610.969318] Hello, world!
[  610.969568] Hello, world!
[  610.969851] hello2 loaded with count=3
```

Результати роботи

- Модулі `hello1.ko` і `hello2.ko` скомпільовано та завантажено під ARM у QEMU.
- Параметр `count` у hello2.ko працює коректно для всіх значень.
- Функція `print_hello()` успішно використовується з іншого модуля.
- Час виконання кожного виклику друку зафіксовано у структурі списку і виведено при видаленні hello1.ko.
- Виявлено, що hello2.ko не можна завантажити без hello1.ko.

Висновки

- Опановано розділення функціоналу між модулями ядра та експорт функцій.
- Навченося працювати зі списками структур і вимірювати час виконання у ядрі.
- Підтверджено роботу параметрів модулів (`module_param`) і взаємозалежність модулів.
- Отримано практичні навички роботи з командами `insmod` , `rmmod` , `modinfo` у середовищі QEMU.

GitHub

GitHub