

CS5250

Assignment 3

Zhang Shuhao
A0120258N

Question 1:

- a. When will **module_init** and **module_exit** be loading/called?
``**lsmod** (\$module).ko'' will load the module named (\$module), and subsequently, **module_init** will be called.
Correspondingly, ``**rmmmod**'' will unload it, and **module_exit** will be called.

- b. What is the command of building the module, installing the module and removing the module?

I write a makefile to support those functions. Specifically, they are implemented as follows.

1. To build the module:

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

2. To install the module:

```
load:
    insmod ./$(TARGET_MODULE).ko
3. To remove the module
unload:
    rmmmod ./$(TARGET_MODULE).ko
```

- c. Give the screenshot of the previous three commands and their results if any in the shell.

The following is the screenshot of ``**dmesg | tail**'' after calling ``**lsmod**'' and ``**rmmmod**'' of the hello world module.

```
shuhaozhang@CS5250:~/Documents/CS5250-OS-$ dmesg | tail
[ 873.035682] raid6: sse2x2 xor() 7364 MB/s
[ 873.083734] raid6: sse2x4 gen() 13259 MB/s
[ 873.131722] raid6: sse2x4 xor() 8912 MB/s
[ 873.131724] raid6: using algorithm sse2x4 gen() 13259 MB/s
[ 873.131724] raid6: .... xor() 8912 MB/s, rmw enabled
[ 873.131725] raid6: using ssse3x2 recovery algorithm
[ 873.134982] xor: automatically using best checksumming function avx
[ 873.155377] Btrfs loaded, crc32c=crc32c-intel
[ 1653.026985] Hello, world
[ 1955.654830] Goodbye, cruel world
shuhaozhang@CS5250:~/Documents/CS5250-OS-$
```

d. Add a <who> parameter to your module so that your module will show hello <who> during init stage.

1. Parameters are declared with the module_param macro, which is defined in moduleparam.h. I highlight the changes in red as follows.

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h> //for module_param
MODULE_LICENSE("GPL");

static char *who = "world";
module_param(who, charp, S_IRUSR|S_IWUSR);
MODULE_PARM_DESC(who, "Tell me your name.");

static int hello_init(void) {
    printk(KERN_ALERT "Hello, %s!\n", who);
    return 0;
}
static void hello_exit(void) {
    printk(KERN_ALERT "Goodbye, %s\n", who);
}

module_init(hello_init);
module_exit(hello_exit);
```

2. lsmod call needs to be updated as well.

```
insmod ./$(TARGET_MODULE).ko who=$(who)
```

I test it with “sudo make load who=“SHUHAOZHANG”, “sudo make unload”. I clean dmesg first by **dmesg -c**. The following screens shot shows the results of **dmesg**.

```
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# sudo make load who="SHUHAOZHANG"
insmod ./hello_world_mod.ko who=SHUHAOZHANG
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# dmesg
[ 3675.906754] Hello, SHUHAOZHANG!
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# make unload
rmmod ./hello_world_mod.ko
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# dmesg
[ 3675.906754] Hello, SHUHAOZHANG!
[ 3684.171161] Goodbye, SHUHAOZHANG!
root@CS5250:/home/shuhaozhang/Documents/CS5250-#
```

Question 2:

- Give the mknod command you use.
`sudo mknod /dev/chardev c 245 0`. Note that, I use dynamic assigning major version instead of fixing at 61, which avoids potential conflict with other device drivers.
- Give the screenshot of your device with “ls -l /dev” command and highlight your device.

```
shuhaozhang@CS5250:~/Documents/CS5250-OS$ ls -l /dev
total 0
crw----- 1 root      root    10, 235 Apr  4 20:01 autofs
drwxr-xr-x 2 root      root    280 Apr  4 20:01 block
drwxr-xr-x 2 root      root    100 Apr  4 20:01 bsg
crw----- 1 root      root    10, 234 Apr  4 20:01 btrfs-control
drwxr-xr-x 3 root      root    60 Apr  4 20:01 bus
[red box highlights this line]
[red box highlights this line]
lrwxrwxrwx 1 root      root    3 Apr  4 20:01 cdrw -> sr0
crw----- 1 root      root    325, 0 Apr  4 20:01 chardev
crw----- 1 root      root    5,  1 Apr  4 20:01 console
lrwxrwxrwx 1 root      root    11 Apr  4 20:01 core -> /proc/kcore
crw----- 1 root      root    10, 59 Apr  4 20:01 cpu_dma_latency
crw----- 1 root      root    10, 203 Apr  4 20:01 cuse
drwxr-xr-x 7 root      root    140 Apr  4 20:01 disk
lrwxrwxrwx 1 root      root    3 Apr  4 20:01 dvd -> sr0
lrwxrwxrwx 1 root      root    3 Apr  4 20:01 dvdrw -> sr0
crw----- 1 root      root    10, 61 Apr  4 20:01 ecryptfs
crw-rw---- 1 root      video   29,  0 Apr  4 20:01 fbo
lrwxrwxrwx 1 root      root    13 Apr  4 20:01 fd -> /proc/self/fd
crw-rw-rw- 1 root      root    1,  7 Apr  4 20:01 full
crw-rw-rw- 1 root      root    10, 229 Apr  4 20:01 fuse
crw----- 1 root      root    254,  0 Apr  4 20:01 gpiochip0
crw----- 1 root      root    10, 228 Apr  4 20:01 heat
```

- Give the codes of read and write functions that you implemented and the screenshots of the four testing cases.

READ:

```
ssize_t onebyte_read(struct file *filep, char *buf, size_t count, loff_t *f_pos)
{
/*please complete the function on your own*/
    int bytes_read = 0;

    /* Check if the buffer has been written */
    if(*buf != 0){
        return 0;
    }
    copy_to_user(buf, onebyte_data, sizeof(char));
    bytes_read++;
    return bytes_read;
}
```

WRITE:

```
ssize_t onebyte_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos)
{
/*please complete the function on your own*/

/**
 * copy_from_user: Returns number of bytes that could not be copied. On success, this will be zero.
 *
 * to
 * Destination address, in kernel space.
 *
 * from
 * Source address, in user space.
 *
```

```

/* Number of bytes to copy.
*/
int bytes_write=0;
copy_from_user(onebyte_data, buf, sizeof(char));

/* Check the length of the bytes that cannot be written*/
if(count> sizeof(char))
{
    printk(KERN_ALERT "No space left on device\n");
    /*Return Linux System Error<28>: No space left on device */
    return -ENOSPC;
}
bytes_write++;
return bytes_write;
}

```

root@CS5250: /home/shuhaozhang/Documents/CS5250-OS-

File Edit View Search Terminal Help

```

[ 2589.500650] 'mknod /dev/chardev c 243 0'.
[ 2589.500651] Try various minor numbers. Try to cat and echo to
[ 2589.500651] the device file.
[ 2589.500651] Remove the device file and module when done.
[ 2589.500652] This is a onebyte device module.
[ 2702.379050] Onebyte device module is unloaded
[ 2817.686648] I was assigned major number 242. To talk to
[ 2817.686650] the driver, create a dev file with
[ 2817.686651] 'mknod /dev/chardev c 242 0'.
[ 2817.686651] Try various minor numbers. Try to cat and echo to
[ 2817.686651] the device file.
[ 2817.686652] Remove the device file and module when done.
[ 2817.686652] This is a onebyte device module.
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# mknod /dev/chardev c 242 0
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-#
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# cat /dev/chardev
Kroot@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# printf a>/dev/chardev
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# cat /dev/chardev
aroot@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# printf b>/dev/chardev
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# cat /dev/chardev
broot@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# printf abc>/dev/chardev
bash: printf: write error: No space left on device
root@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# cat /dev/chardev
aroot@CS5250:/home/shuhaozhang/Documents/CS5250-OS-# 
```

The following is the screenshot of github commit. The last two commit is made inside the virtual machine.

Commits on Apr 4, 2018
read and write function implemented shuhaozhang committed 4 minutes ago
add simple return to read and write function shuhaozhang committed an hour ago
cat receive bad address error, so try to debug. ShuhaoZhangTony committed 4 hours ago
rename file. ShuhaoZhangTony committed 4 hours ago
update device driver with new name and dynamic binding major version ShuhaoZhangTony committed 4 hours ago
amend ShuhaoZhangTony committed 7 hours ago
update makefile ShuhaoZhangTony committed 7 hours ago
add initial version of device module. ShuhaoZhangTony committed 7 hours ago
amend. ShuhaoZhangTony committed 8 hours ago
update Makefile accordingly ShuhaoZhangTony committed 8 hours ago
add <who> parameter ShuhaoZhangTony committed 8 hours ago
``tab`` problem ShuhaoZhangTony committed 9 hours ago

APPENDIX

The following are the source code I used in this project, I have highlighted the changes or important code lines in red.

Chardev.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/uaccess.h>
#define MAJOR_NUMBER 61/* forward declaration */

static int Major; /* Major number assigned to our device driver */
#define DEVICE_NAME "chardev" /* Dev name as it appears in /proc/devices */
```

```

int onebyte_open(struct inode *inode, struct file *filep);
int onebyte_release(struct inode *inode, struct file *filep);
ssize_t onebyte_read(struct file *filep,
                     char *buf, size_t count,
                     loff_t *f_pos);

ssize_t onebyte_write(struct file *filep,
                      const char *buf, size_t count,
                      loff_t *f_pos);

static void onebyte_exit(void);
/* definition of file_operation structure */
struct file_operations onebyte_fops = {
    read: onebyte_read,
    write: onebyte_write,
    open: onebyte_open,
    release: onebyte_release
};
char *onebyte_data = NULL;

/*
 * Called when a process tries to open the device file, like
 * "cat /dev/mycharfile"
 */
int onebyte_open(struct inode *inode, struct file *filep)
{
    return 0; // always successful
}
int onebyte_release(struct inode *inode, struct file *filep)
{
    return 0; // always successful
}
ssize_t onebyte_read(struct file *filep, char *buf, size_t count, loff_t *f_pos)
{
/*please complete the function on your own*/
    int bytes_read = 0;
    /* Check if the buffer has been written */
    if(*buf != 0){
        return 0;
    }
    copy_to_user(buf, onebyte_data, sizeof(char));
    bytes_read++;
    return bytes_read;
}
ssize_t onebyte_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos)

```

```

{
/*please complete the function on your own*/

/*
 * copy_from_user: Returns number of bytes that could not be copied. On success, this will
be zero.
*
* to
* Destination address, in kernel space.
*
* from
* Source address, in user space.
*
* n
* Number of bytes to copy.
*/
int bytes_write=0;
copy_from_user(onebyte_data, buf, sizeof(char));

/* Check the length of the bytes that cannot be written*/
if(count> sizeof(char))
{
    printk(KERN_ALERT "No space left on device\n");
    /*Return Linux System Error<28>: No space left on device */
    return -ENOSPC;
}
bytes_write++;
return bytes_write;

}

static int onebyte_init(void)
{
// int result;
// register the device
Major = register_chrdev(0, DEVICE_NAME, &onebyte_fops);
if (Major < 0) {
    printk(KERN_ALERT "Registering char device failed with %d\n", Major);
    return Major;
}
printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
printk(KERN_INFO "the driver, create a dev file with\n");
printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
printk(KERN_INFO "the device file.\n");
printk(KERN_INFO "Remove the device file and module when done.\n");

// allocate one byte of memory for storage

```

```

// kmalloc is just like malloc, the second parameter is// the type of memory to be
allocated.
// To release the memory allocated by kmalloc, use kfree.
onebyte_data = kmalloc(sizeof(char), GFP_KERNEL);
if (!onebyte_data) {
    onebyte_exit();
    // cannot allocate memory
    // return no memory error, negative signify a failure
    return -ENOMEM;
}
// // initialize the value to be X
*onebyte_data = 'X';
printk(KERN_ALERT "This is a onebyte device module.\n");
return 0;
}
static void onebyte_exit(void)
{
    // if the pointer is pointing to something
    if (onebyte_data) {
        // free the memory and assign the pointer to NULL
        kfree(onebyte_data);
        onebyte_data = NULL;
    }
    // unregister the device
    unregister_chrdev(MAJOR_NUMBER, DEVICE_NAME);
    printk(KERN_ALERT "Onebyte device module is unloaded\n");
}
MODULE_LICENSE("GPL");
module_init(onebyte_init);
module_exit(onebyte_exit);

```

Makefile

```

TARGET_MODULE:=chardev
obj-m += $(TARGET_MODULE).o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
load:
    insmod ./$(TARGET_MODULE).ko
unload:
    rmmod ./$(TARGET_MODULE).ko

```

Hello_world_mod

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

```

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

#include <linux/moduleparam.h> //for module_param
MODULE_LICENSE("GPL");

static char *who = "world";
module_param(who, charp, S_IRUSR|S_IWUSR);
MODULE_PARM_DESC(who, "Tell me your name.");

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, %s!\n", who);
    return 0;
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, %s!\n", who);
}
module_init(hello_init);
module_exit(hello_exit);

```

Makefile

```

TARGET_MODULE:= hello_world_mod
obj-m += $(TARGET_MODULE).o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
load:
    insmod ./$(TARGET_MODULE).ko who=($who)
unload:
    rmmod ./$(TARGET_MODULE).ko

```