

Kennesaw State University

CSE 3502 Operating Systems

Project - The /Proc File Systems and mmap

Assignment 1: Create a helloworld kernel module (20 pts)

The following code is a complete helloworld module. Name it as ***new_module.c***

<https://github.com/kevinsuo/CS3502/blob/master/project-4-1.c>

```
-----  
#include <linux/module.h>  
#include <linux/kernel.h>  
  
int init_new_module(void)  
{  
    printk(KERN_INFO "Hello, world!\n");  
    return 0;  
}  
  
void exit_new_module(void) {  
    printk(KERN_INFO "Goodbye, world!\n");  
}  
  
module_init(init_new_module);  
module_exit(exit_new_module);  
-----
```

The module defines two functions. `init_module` is invoked when the module is loaded into the kernel and `exit_module` is called when the module is removed from the kernel. `module_init` and `module_exit` are special kernel macros to indicate the role of these two functions.

Use the following makefile to compile the module. Name it as ***Makefile***

<https://github.com/kevinsuo/CS3502/blob/master/project-4-1-Makefile>

Note that here ***new_module.o*** is the output after compiling.

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

Compile the `new_module.c` file using `make` command.

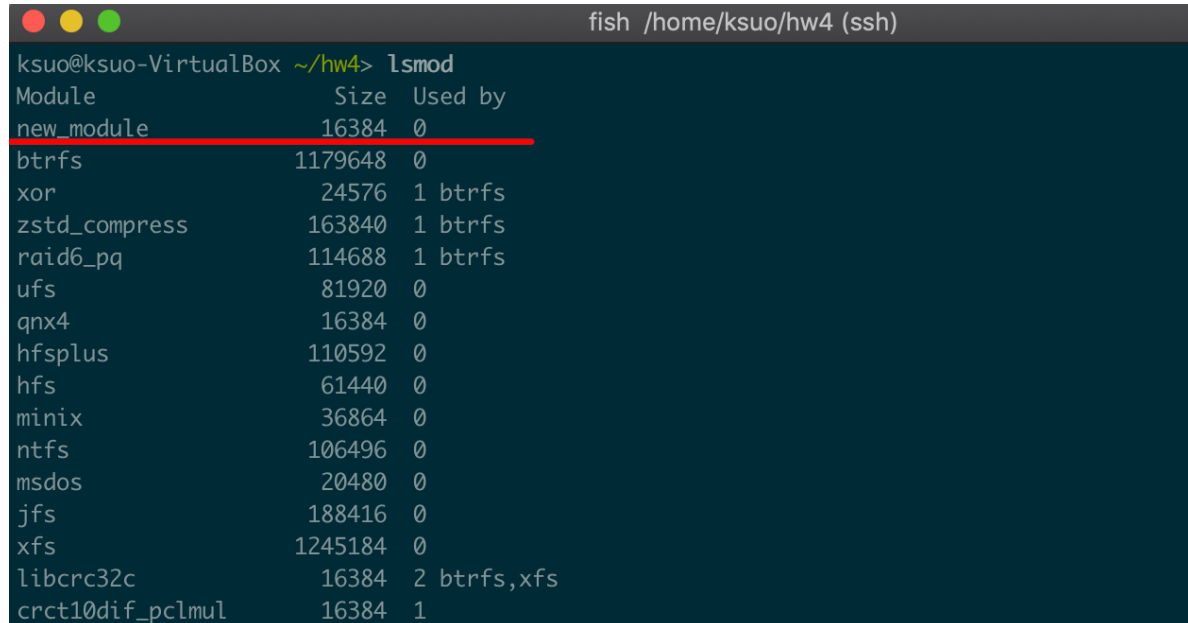
make

To insert the module into the Linux kernel:

```
# sudo insmod new_module.ko
```

Use the following command to verify the module has been loaded:

```
# lsmod
```

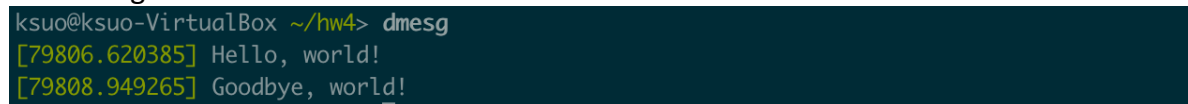


Module	Size	Used by
new_module	16384	0
btrfs	1179648	0
xor	24576	1 btrfs
zstd_compress	163840	1 btrfs
raid6_pq	114688	1 btrfs
ufs	81920	0
qnx4	16384	0
hfsplus	110592	0
hfs	61440	0
minix	36864	0
ntfs	106496	0
msdos	20480	0
jfs	188416	0
xfs	1245184	0
libcrc32c	16384	2 btrfs,xfs
crct10dif_pclmul	16384	1

To remove the module from the kernel:

```
# sudo rmmod new_module
```

When you insert or remove the module, corresponding information will be printed out under the dmesg.



```
[79806.620385] Hello, world!  
[79808.949265] Goodbye, world!
```

Assignment 2: Create an entry in the /proc file system for user level read and write (30 pts)

Write a new kernel module following steps in assignment 1. This module creates an entry in the /proc file system. Use the following code skeleton to write the module:

<https://github.com/kevinsuo/CS3502/blob/master/project-4-2.c>

```

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/vmalloc.h>
#include <linux/slab.h>
#include <linux/uaccess.h>

#define MAX_LEN      4096
static struct proc_dir_entry *proc_entry;

ssize_t read_proc(struct file *f, char *user_buf, size_t count, loff_t *off)
{
    //output the content of info to user's buffer pointed by page
    return count;
}

ssize_t write_proc(struct file *f, const char *user_buf, size_t count, loff_t *off)
{
    //copy the written data from user space and save it in info
    return count;
}

struct file_operations proc_fops = {
    .read = read_proc,
    .write = write_proc
};

int init_module( void )
{
    int ret = 0;
    //create the entry named myproc and allocated memory space for the proc entry

    printk(KERN_INFO "test_proc created.\n");

    return ret;
}

void cleanup_module( void )
{
    //remove the entry named myproc and free info space
}

```

Step 1: create an entry in proc file system named *myproc* when the kernel module is loaded; this entry *myproc* will be deleted when the kernel mode is deleted. You can use `$ ls /proc/` to check whether it is existed. (Hint: `proc_create()` and `remove_proc_entry()` are needed.)

Step 2: implement `read_proc` and `write_proc` function to read/write the proc file entry in Step 1. You need to add codes for allocating memory in `init_module` and releasing the memory in `cleanup_module` for the proc file entry. (Hint: `copy_to_user()` is needed for the read and `copy_from_user()` is needed for write.)

To test your results, load the kernel module and there should be a new entry created under `/proc`. Use `cat` and `echo` to verify and change the content of the new entry.

```
ksuo@ksuo-VirtualBox ~/hw4-2> sudo insmod my_proc.ko
[sudo] password for ksu0:
ksuo@ksuo-VirtualBox ~/hw4-2> ls /proc/
1/      1283/ 1471/ 23/    39/    497/ 683/    diskstats  pagetypeinfo
10/     1284/ 15/    24/    4/     50/ 686/    dma        partitions
11/     1285/ 1502/ 249/   40/    500/ 7/      driver/    pressure/
1114/   1287/ 1504/ 250/   41/    502/ 702/    execdomains sched_debug
1119/   1288/ 1522/ 251/   42/    503/ 748/    fb         schedstat
1124/   1295/ 154/   254/   423/   506/ 8/      filesystems scsi/
1137/   13/    155/   26/    43/    509/ 803/    fs/        self@
1142/   1303/ 156/   27/    438/   52/ 804/    interrupts slabinfo
1144/   1304/ 157/   275/   44/    523/ 817/    iomem      softirqs
1168/   1314/ 158/   276/   440/   53/ 821/    ioports    stat
1174/   1315/ 159/   28/    449/   532/ 823/    irq/       swaps
1189/   1321/ 1598/ 282/   45/    533/ 891/    kallsyms   sys/
1190/   1323/ 16/    29/    453/   534/ 9/      kcore      sysrq-trigger
12/     1325/ 161/   292/   454/   537/ 905/    keys       sysvipc/
1205/   1327/ 162/   3/     455/   54/ 912/    key-users  thread-self@
1209/   1331/ 1684/ 30/    457/   56/ 931/    kmsg       timer_list
1210/   1332/ 1685/ 32/    46/    571/ 950/    kpagecgroup tty/
1212/   1337/ 1695/ 321/   47/    572/ 954/    kpagecount uptime
1218/   1338/ 17/    33/    48/    59/ 975/    kpageflags version
1229/   1372/ 173/   331/ 482/   6/  acpi/    loadavg    vmallocinfo
1233/   1383/ 1763/ 335/   484/   60/  asound/   locks      vmstat
1241/   1384/ 18/    336/ 485/   606/  buddyinfo mdstat     zoneinfo
1245/   14/    19/    34/    489/   607/  bus/     meminfo
1252/   1412/ 192/   35/    49/    608/  cgroups  misc
1261/   1415/ 193/   36/    490/   61/  cmdline  modules
1266/   1439/ 2/     360/   491/   614/  consoles mounts@
1271/   1442/ 20/    37/    493/   634/  cpuinfo  mtrr
1275/   1446/ 21/    38/    494/   659/  crypto   myproc
1279/   1460/ 22/    384/ 496/   677/  devices  net@
```

You can use the following to test the read or write on the entry of proc file system. Here the root user is needed.

```
root@ksuo-VirtualBox /h/k/hw4-2# echo 12345 > /proc/myproc
root@ksuo-VirtualBox /h/k/hw4-2# cat /proc/myproc
12345
```

Assignment 3: Exchange data between the user and kernel space via mmap (50 pts)

Write a kernel module that create an entry in the /proc file system. The new entry cannot be directly read or written using cat and echo commands. Instead, map the new entry to a user space memory area so that user-level processes can read from and write to the kernel space via mmap. The skeleton of the kernel module is given below:

<https://github.com/kevinsuo/CS3502/blob/master/project-4-3-1.c>

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

```

#include <linux/types.h>
#include <linux/kthread.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/slab.h>

static struct proc_dir_entry *tempdir, *tempinfo;
static unsigned char *buffer;
static unsigned char array[12]={0,1,2,3,4,5,6,7,8,9,10,11};

static void allocate_memory(void);
static void clear_memory(void);
static int my_map(struct file *filp, struct vm_area_struct *vma);

static const struct file_operations myproc_fops = {
    .mmap = my_map,
};

static int my_map(struct file *filp, struct vm_area_struct *vma)
{
    // map vma of user space to a continuous physical space
    return 0;
}

static int init_myproc_module(void)
{
    tempdir=proc_mkdir("mydir", NULL);
    if(tempdir == NULL) {
        printk("mydir is NULL\n");
        return -ENOMEM;
    }

    tempinfo = proc_create("myinfo", 0, tempdir, &myproc_fops);
    if(tempinfo == NULL) {
        printk("myinfo is NULL\n");
        remove_proc_entry("mydir",NULL);
        return -ENOMEM;
    }
    printk("init myproc module successfully\n");

    allocate_memory();

    return 0;
}

static void allocate_memory(void)
{
    /* allocation memory */
    buffer = (unsigned char *)kmalloc(PAGE_SIZE,GFP_KERNEL);
    /* set the memory as reserved */
    SetPageReserved(virt_to_page(buffer));
}

static void clear_memory(void)
{
    /* clear reserved memory */
    ClearPageReserved(virt_to_page(buffer));
    /* free memory */
    kfree(buffer);
}

static void exit_myproc_module(void)
{
    clear_memory();
    remove_proc_entry("myinfo", tempdir);
    remove_proc_entry("mydir", NULL);
    printk("remove myproc module successfully\n");
}

```

```

module_init(init_myproc_module);
module_exit(exit_myproc_module);
MODULE_LICENSE("GPL");

```

The above code will create an entry `/proc/mydir/myinfo` under the proc file system. You are required to implement the ***my_map*** function to map one piece of memory (***char array[12]***) into user space. Then write a user space program using `mmap` to visit the memory space of the proc file and print the data in that memory area. You can use the following skeleton: <https://github.com/kevinsuo/CS3502/blob/master/project-4-3-2.c>

```

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#define PAGE_SIZE 4096

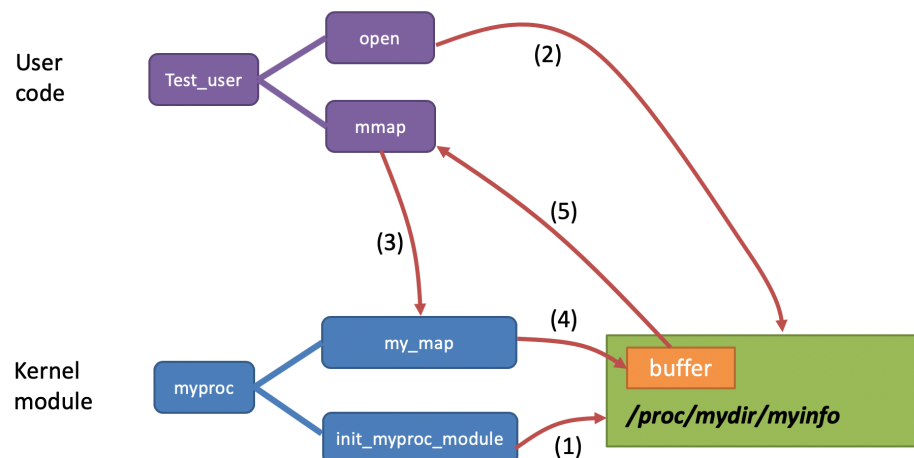
int main(int argc , char *argv[])
{
    int fd;
    int i;
    unsigned char *p_map;

    /* open proc file */
    fd = open("/proc/mydir/myinfo", O_RDWR);
    if(fd < 0) {
        printf("open fail\n");
        exit(1);
    } else {
        printf("open successfully\n");
    }

    // map p_map to the proc file and grant read & write privilege
    // read data from p_map
    // unmap p_map from the proc file

    return 0;
}

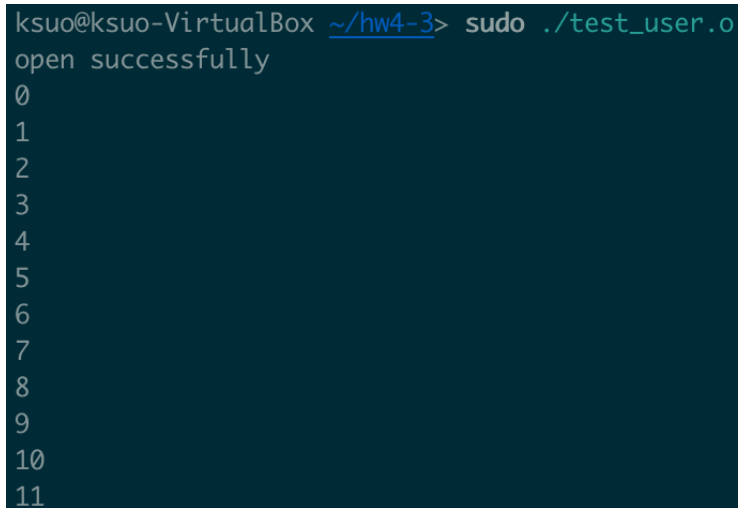
```



The above figure shows the entire workflow:

- (1) Kernel module create a proc file: ***/proc/mydir/myinfo***
- (2) User process open the created proc file
- (3) User process calls mmap function, which further executed my_map defined in the kernel
- (4) my_map() then maps one piece of memory into user space (e.g., buffer) and puts some data inside
- (5) User process visits this piece of memory and prints the data out.

Expected output:

A terminal window with a dark blue background. The prompt is 'ksuo@ksuo-VirtualBox ~/hw4-3>'. The command 'sudo ./test_user.o' has been entered. The output is 'open successfully' followed by a vertical list of integers from 0 to 11.

```
ksuo@ksuo-VirtualBox ~/hw4-3> sudo ./test_user.o
open successfully
0
1
2
3
4
5
6
7
8
9
10
11
```

Submission requirements:

Create one folder and put **your modification (file diff1.txt, diff2.txt, ...)** and **the screenshot of output** into this folder. Please use diff command to highlight your modification:

```
$ diff -u original_file.c modified_file.c > diff.txt
```

Upload the code and screenshot file onto D2L.