**Kennesaw State University**

**CSE 3502 Operating Systems**

**Project - The /Proc File Systems and mmap**

**Part 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

A screenshot of a cell phone

Description automatically generated

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.

A close up of a logo

Description automatically generated

**Part 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 Part 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.

A screenshot of a computer

Description automatically generated

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

Write 12345 into /proc/myproc

A close up of a logo

Description automatically generated

Read /proc/myproc and printout its content:

A close up of a logo

Description automatically generated

**Part 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**;

}

--------------------------------------------------------------------------------------------------------------------------

A close up of a map

Description automatically generated

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 close up of a logo

Description automatically generated

**Submission requirements:**

Submit your assignment file through D2L using the appropriate link.

The submission must include the ***source code***, ***output screenshot of your code*** and ***a report describe your code logic***.