

Project 4 - /proc File Systems and mmap

Total Points: 100, plus 40 bonus points

Out: Nov 12, 2018

Due: 11:59 pm, Tuesday, Nov 27, 2018

Introduction

The purpose of this project is to practice writing Linux kernel modules to create a new device file and an entry in the `proc` file system. The objectives of this project is to learn:

1. How to write a helloworld Linux kernel module.
2. How to create a new device in Linux.
3. How to add a new entry in the `proc` file system.

Project submission

For each project, please create a zipped file containing the following items, and submit it to Blackboard.

1. A report that includes the name of your virtual machine (VM), and the password for the `instructor` and `root` accounts (details see *Accessing VM* below); and (3) a brief description about how you solved the problems and what you learned. The report can be in `txt`, `doc`, or `pdf` format.
2. Your code.

Accessing VM. Each team should specify the name of your VM that the instructor can login to check the project results. Please provide the **password for root and instructor** (note this is different from the last 3 projects).

Project

Part 1: Create a Helloworld kernel module (30 points)

The following code is a complete helloworld module.

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

int init_module(void) {
    printk (KERN_INFO "Hello world!\n");
    return 0;
}

void cleanup_module(void){
    printk (KERN_INFO "Goodbye world!\n");
}

module_init(init_module);
module_exit(cleanup_module);

MODULE_LICENSE("GPL");
```

The module defines two functions. `init_module` is invoked when the module is loaded into the kernel and `cleanup_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 (you need **root** permission).

```
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
```

Do you see any errors? If so, how to fix the error?

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
```

To remove the module from the kernel:

```
# sudo rmmod new_module
```

Use `dmesg` to see the module's output.

Part 2: Create an entry in the /proc file system for user level read and write (70 points)

Write a kernel module that creates an entry in the `/proc` file system. Use the following code skeleton to write the module:

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

```

#define MAXLEN 4096
int read_info(char *page, char **start, off_t off, int count, int *eof,
              void *data);
ssize_t write_info(struct file *filp, const char __user *buff, unsigned
                  long len, void *data);

static struct proc_dir_entry *proc_entry;
static char *info;
static int write_index;
static int read_index;

int init_module(void) {
    int ret = 0;
    // allocated memory space for the proc entry
    info = (char *)vmalloc(MAXLEN);
    memset(info, 0, MAXLEN);

    //implement this: create the proc entry

    write_index = 0;
    read_index = 0;

    //register the write and read callback functions
    proc_entry->read_proc = read_info;
    proc_entry->write_proc = write_info;
    printk(KERN_INFO "test_proc created .\n");
    return ret;
}

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

ssize_t write_info(struct file *filp, const char __user *buff,
                  unsigned long len, void * data){
    //copy the written data from user space and save it in info
    return len;
}

int read_info(char *buff, char **start, off_t offset, int count,
              int *eof, void *data){
    //output the content of info to user's buffer pointed by buff
    return len;
}

```

The callback functions `write_info` and `read_info` will be invoked whenever the `proc` file is written and read, respectively, e.g., using the `cat` and `echo` commands. `write_info` uses the `copy_from_user` function to communicate with the user space. 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.

Note: you need to change the previous Makefile's `obj-m += new_module.o` line to compile the module.

Part 3: Exchange data between the user and kernel space via mmap (bonus 40 points)

Write a kernel module that creates an entry in the `/proc` file system. The new entry can not be directly read from or written to using `cat` and `echo`. 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:

```
#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>
#include <asm/io.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);
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 *flip, struct vm_area_struct *vma){
    // map vma of user space to a continuous physical space
    return 0;
}

static int init_myproc_module(void){
    //create a directory in /proc
    //create a new entry under the new directory
    printk("init myproc module successfully\n");

    allocate_memory();
    //initialize the buffer
    for(i=0; i<12; i++){
        buffer[i] = array[i];
    }

    return 0;
}

static void allocate_memory(void){
    //allocation memory
```

```

    //set the memory as reserved
}

static void clear_memory(void){
    //clear reserved memory
    //free memory
}

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");

```

Write a user space program to test the `proc` file you just created. Use the following skeleton:

```

#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[]) {
    unsigned char *p_map ;
    // open proc file
    // map p_map to the proc file and grant read & write privilege
    // design test case to read from and write to p_map
    // unmap p_map from the proc file
    return 0 ;
}

```