



Факультет программной инженерии и компьютерной техники
Направление подготовки: Информатика и вычислительная техника
Дисциплина «Операционные системы»

Лабораторная работа №2

Вариант IOCTL: Ismod, fpu_state

Выполнил:

Богатов А. С.

P33302

Преподаватель

Осипов С.В.

г. Санкт-Петербург, 2022 г.

Задание:

Разработать комплекс программ на пользовательском уровне и уровне ядра, который собирает информацию на стороне ядра и передает информацию на уровень пользователя, и выводит ее в удобном для чтения человеком виде. Программа на уровне пользователя получает на вход аргумент(ы) командной строки (не адрес!), позволяющие идентифицировать из системных таблиц необходимый путь до целевой структуры, осуществляет передачу на уровень ядра, получает информацию из данной структуры и распечатывает структуру в стандартный вывод. Загружаемый модуль ядра принимает запрос через указанный в задании интерфейс, определяет путь до целевой структуры по переданному запросу и возвращает результат на уровень пользователя.

Интерфейс передачи между программой пользователя и ядром и целевая структура задается преподавателем. Интерфейс передачи может быть один из следующих:

1. `syscall` - интерфейс системных вызовов.
2. `ioctl` - передача параметров через управляющий вызов к файлу/устройству.
3. `procfs` - файловая система `/proc`, передача параметров через запись в файл.
4. `debugfs` - отладочная файловая система `/sys/kernel/debug`, передача параметров через запись в файл.

Целевая структура может быть задана двумя способами:

1. Именем структуры в заголовочных файлах Linux
2. Файлом в каталоге `/proc`. В этом случае необходимо определить целевую структуру по пути файла в `/proc` и выводимым данным.

Makefile:

```
obj-m += driver.o
KDIR = /lib/modules/$(shell uname -r)/build
KBUILD_CFLAGS += -g -Wall
all:
    make -C $(KDIR) M=$(shell pwd) modules
clean:
    make -C $(KDIR) M=$(shell pwd) clean
```

Драйвер:

```
#ifndef __IOCTL_H
```

```

#define __IOCTL_H

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

struct user_fpu_state {
    uint16_t control_word;
    uint16_t state_word;
    uint16_t last_instr_opcode;
    uint64_t instr_pointer;
    uint64_t data_pointer;
    uint32_t ip_offset;
    uint32_t ip_selector;
    uint32_t operand_offset;
    uint32_t operand_selector;
    struct fpstate* fpu_state_;
};

struct user_lsmod {
    char* module_names;
    struct user_lsmod* next;
    struct module* module_;
};

#define IOCTL_BASE 'i'
#define WR_VALUE _IOW(IOCTL_BASE,1,int32_t*)
#define RD_LSMOD _IOR(IOCTL_BASE,2, struct module*)
#define RD_FPU_STATE _IOR(IOCTL_BASE,3, struct user_fpu_state*)
#define IOCTL_GET_MSG _IOR(IOCTL_BASE, 4, char *)

#endif

```

Задаем структуры для передачи на уровень пользователя и методы.

driver.c

```

/*****
*****//**
* \file      driver.c
*
* \details   Lsmod & FPU State Linux device driver (IOCTL)

```

```

*
*  \author      abogatov
*
*****
*****/
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/init.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/err.h>
#include <linux/pid.h>
#include <asm/uaccess.h>
#include <linux/pid_namespace.h>
#include <linux/string.h>
#include <linux/ctype.h>
#include "driver.h"
#define BUFFER 4096
static char lsmmod[BUFFER];
static char* lsmmod_ptr;
static char* lsmmod_ptr_current;
struct task_struct* task;
struct thread_struct* threadp;
struct pid* pid;
struct fpu* fpu_kernel;
int32_t value = 0;
dev_t dev = 0;
static struct class *dev_class;
static struct cdev driver_cdev;
/*
** Function Prototypes
*/
static int      __init driver_driver_init(void);
static void     __exit driver_driver_exit(void);
static int      driver_open(struct inode *inode, struct file
*file);
static int      driver_release(struct inode *inode, struct file
*file);

```

```

static ssize_t driver_read(struct file *filp, char __user *buf,
size_t len,loff_t * off);
static ssize_t driver_write(struct file *filp, const char *buf,
size_t len, loff_t * off);
static long driver_ioctl(struct file *file, unsigned int cmd,
unsigned long arg);
/*
** File operation sturcture
*/
static struct file_operations fops =
{
    .owner          = THIS_MODULE,
    .read           = driver_read,
    .write          = driver_write,
    .open           = driver_open,
    .unlocked_ioctl = driver_ioctl,
    .release        = driver_release,
};
/*
** This function will be called when we open the Device file
*/
static int driver_open(struct inode *inode, struct file *file)
{
    pr_info("Device File Opened...!!!\n");
    return 0;
}
/*
** This function will be called when we close the Device file
*/
static int driver_release(struct inode *inode, struct file *file)
{
    pr_info("Device File Closed...!!!\n");
    return 0;
}
/*
** This function will be called when we read the Device file
*/
static ssize_t driver_read(struct file *filp, char __user *buf,
size_t length, loff_t *off)
{
    pr_info("Read Function\n");
    if (*lsmod_ptr == 0) {

```

```

        pr_info("nothing to transmit%s", lsmode_ptr);
        return 0;
    }

    while (length && *lsmode_ptr) {

        put_user(*(lsmode_ptr++), buf++);
        length--;

    }

    return 0;
}
/*
** This function will be called when we write the Device file
*/
static ssize_t driver_write(struct file *filp, const char __user
*buf, size_t len, loff_t *off)
{
    pr_info("Write function\n");
    return len;
}
/*
** This function will be called when we write IOCTL on the Device
file
*/
static long driver_ioctl(struct file *file, unsigned int cmd,
unsigned long arg)
{
    switch(cmd) {
        case WR_VALUE:
            if (copy_from_user(&value, (int32_t*) arg,
sizeof(value))) {
                pr_err("Data Write: Err\n");
            }
            break;
        case IOCTL_GET_MSG:
            driver_read(file, (char*) arg, strlen(lsmode), 0);
            break;
        case RD_LSMODE:
            char* ptr;
            struct file *f1;

```

```

struct file *f2;
char* file_name = kmalloc(32, GFP_KERNEL);
char buff1[32];
int i;
for (i = 0; i < 32; i++)
    buff1[i] = 0;
char buff2[32];
for (i = 0; i < 32; i++)
    buff2[i] = 0;
struct user_lsmod begin = (struct user_lsmod) {
    .module_names = THIS_MODULE->name,
    .module_ = THIS_MODULE,
    .next = NULL
};
struct user_lsmod begin2 = (struct user_lsmod) {
    .module_names = "",
    .next = NULL
};
struct user_lsmod* user_modules = &begin;
struct user_lsmod* next_mod = &begin2;
struct list_head* list;
struct module* mod;
struct module* this_mod = THIS_MODULE;
pr_info("%-18s %-10s %-3s", "Module", "Size",
"Used by");

lsmod_ptr_current += sprintf(lsmod_ptr_current,
"%-18s %-10s %-3s\n", "Module", "Size", "Used by");
list_for_each(list, &this_mod->list) {
    if(list_entry(list, struct module, list) !=
NULL)

    mod = list_entry(list, struct module, list);
    next_mod->module_names = mod->name;
    if (strcmp(mod->name, "") != 0) {
        user_modules->next = next_mod;
        strcpy(file_name, "/sys/module/");
        strcat(file_name, mod->name);
        strcat(file_name, "/coresize");
        f1 = filp_open(file_name, O_RDONLY, 0);
        if (f1 == NULL)
            pr_err("Failed to open file");
        else {

```

```

        kernel_read(f1, buff1, 16, &f1->f_pos);

        filp_close(f1, NULL);
        for (ptr = buff1 + strlen(buff1) - 1;
(ptr >= buff1) && isspace(*ptr); --ptr);
        ptr[1] = '\\0';
    }

    strcpy(file_name, "/sys/module/");
    strcat(file_name, mod->name);
    strcat(file_name, "/refcnt");
    f2 = filp_open(file_name, O_RDONLY, 0);
    if (f2 == NULL)
        pr_err("Failed to open file");
    else {

        kernel_read(f2, buff2, 16, &f2->f_pos);

        filp_close(f2, NULL);
        for (ptr = buff2 + strlen(buff2) - 1;
(ptr >= buff2) && isspace(*ptr); --ptr);
        ptr[1] = '\\0';
    }

    pr_info("%-18s %-10s %-3s", mod->name,
buff1, buff2);

    lsmod_ptr_current +=
sprintf(lsmod_ptr_current, "%-18s %-10s %-3s\\n", mod->name, buff1,
buff2);

    next_mod->next = NULL;
    next_mod->module_ = mod;
    user_modules = next_mod;
}
}
pr_info("Here we are!\\n");
pr_info("We recorded first module %s\\n",
begin.module_names);

begin.lsmod_len = strlen(lsmod);
if(copy_to_user((struct user_lsmod*) arg, &begin,
sizeof(begin)))
{

```



```

pr_err("Data Write1 :
Err!\n");

    }

    break;
case RD_FPU_STATE:
pr_info("Well it's started");
threadp = kmalloc(sizeof(struct thread_struct),
GFP_KERNEL);
fpu_kernel = kmalloc(sizeof(struct fpu),
GFP_KERNEL);

pid = find_get_pid((int) value);
task = pid_task(pid, PIDTYPE_PID);
threadp = &(task->thread);
fpu_kernel = &(threadp->fpu);
struct fpstate* fpu_state_kernel =
kmalloc(sizeof(struct fpstate), GFP_KERNEL);
fpu_state_kernel = fpu_kernel->fpstate;
struct user_fpu_state user_fpstate = {
    .control_word =
fpu_state_kernel->regs.xsave.i387.cwd,
    .state_word =
fpu_state_kernel->regs.xsave.i387.swd,
    .last_instr_opcode =
fpu_state_kernel->regs.xsave.i387.fop,
    .instr_pointer =
fpu_state_kernel->regs.xsave.i387.rip,
    .data_pointer =
fpu_state_kernel->regs.xsave.i387.rdp,
    .ip_offset =
fpu_state_kernel->regs.xsave.i387.fip,
    .ip_selector =
fpu_state_kernel->regs.xsave.i387.fcs,
    .operand_offset =
fpu_state_kernel->regs.xsave.i387.foo,
    .operand_selector =
fpu_state_kernel->regs.xsave.i387.fos,
    .fpu_state_ = fpu_state_kernel
};
pr_info("Copy init");
if(copy_to_user((struct user_fpu_state *)
arg, &user_fpstate, sizeof(user_fpstate)) )

```

```

        {
            pr_err("Data Read : Err!\n");
        }
        pr_info("Copy finished");
        break;
    default:
        pr_info("Default\n");
        break;
    }
    return 0;
}
/*
** Module Init function
*/
static int __init driver_driver_init(void)
{
    /*Allocating Major number*/
    if((alloc_chrdev_region(&dev, 0, 1, "driver_Dev")) < 0){
        pr_err("Cannot allocate major number\n");
        return -1;
    }
    pr_info("Major = %d Minor = %d \n", MAJOR(dev),
MINOR(dev));
    /*Creating cdev structure*/
    cdev_init(&driver_cdev, &fops);
    /*Adding character device to the system*/
    if((cdev_add(&driver_cdev, dev, 1)) < 0){
        pr_err("Cannot add the device to the system\n");
        goto r_class;
    }
    /*Creating struct class*/
    if(IS_ERR(dev_class =
class_create(THIS_MODULE, "driver_class"))){
        pr_err("Cannot create the struct class\n");
        goto r_class;
    }
    /*Creating device*/

    if(IS_ERR(device_create(dev_class, NULL, dev, NULL, "driver_device"))){
        pr_err("Cannot create the Device 1\n");
        goto r_device;
    }
}

```

```

    }
    pr_info("Device Driver Insert...Done!!!\n");
    int i;
    for (i = 0; i < BUFFER; i++)
        lsmode[i] = "\0";
    lsmode_ptr = lsmode;
    lsmode_ptr_current = lsmode;
    return 0;
r_device:
    class_destroy(dev_class);
r_class:
    unregister_chrdev_region(dev,1);
    return -1;
}
/*
** Module exit function
*/
static void __exit driver_driver_exit(void)
{
    device_destroy(dev_class,dev);
    class_destroy(dev_class);
    cdev_del(&driver_cdev);
    unregister_chrdev_region(dev, 1);
    pr_info("Device Driver Remove...Done!!!\n");
}
module_init(driver_driver_init);
module_exit(driver_driver_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("abogatov");
MODULE_DESCRIPTION("Lsmode & FPU State Linux device driver
(IOCTL)");
MODULE_VERSION("1.5");

```

Определяем методы инициализации и уничтожения модуля, операции чтения/записи и метод через который передаем данные на уровень пользователя.

test_app.c

```

#include <stdio.h>
#include <stdlib.h>

```

```

#include <string.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/sched.h>
#include "driver.h"
#define STRUCT(type, pInstance, ...)
\
{
\
    printf("%s=%p: {\n", #type, &pInstance);
\
    type *pStr = &pInstance;
\
    __VA_ARGS__
\
    printf("}\n");
\
}
#define FIELD(pPat, pInstance)
\
{ printf("  %s=%" #pPat "\n", #pInstance, pStr->pInstance); }
// for inttypes...
#define PFIELD(pPat, pInstance)
\
{ printf("  %s=%" pPat "\n", #pInstance, pStr->pInstance); }
int main()
{
    int fd;
    int32_t number;
    int32_t value;
    char message[4096];
    int i;
    for (i = 0; i < 4096; i++)
        message[i] = "\0";
    printf("\nOpening Driver\n");
    fd = open("/dev/driver_device", O_RDWR);
    if (fd < 0) {
        printf("Cannot open device file...\n");
    }
}

```

```

        return 0;
    }
    number = getpid();
    printf("Writing Value to Driver\n");
    ioctl(fd, WR_VALUE, (int32_t*) &number);
    printf("Reading Value from Driver\n");
    struct user_lsmod user_modules = (struct user_lsmod) {
        .module_names = ""
    };
    if (ioctl(fd, RD_LSMOD, &user_modules.module_names) < 0) {
        printf("oopsie");
    } else {
        //printf("For lsmod output please see
dmsg\nPointer:\n");
        STRUCT (struct user_lsmod, user_modules, FIELD(p,
module_names) FIELD(p, module_) FIELD(p, next) FIELD(u,
lsmod_len));
        ioctl(fd, IOCTL_GET_MSG, message);
        printf("%s", message);
    }
    struct user_fpu_state user_fpstate;
    if (ioctl(fd, RD_FPU_STATE, &user_fpstate) < 0)
        printf("oopsie");
    else {
        STRUCT (struct user_fpu_state, user_fpstate,
FIELD(x, control_word)
                FIELD(x, state_word) FIELD(x,
last_instr_opcode) FIELD(x, instr_pointer) FIELD(x, data_pointer)
                FIELD(x, ip_offset) FIELD(x,
ip_selector) FIELD(x, operand_offset) FIELD(x, operand_selector)
FIELD(p, fpu_state_));
    }
    printf("Closing Driver\n");
    close(fd);
}

```

Для достижения структуры fpu_state_struct используем pid текущего процесса, далее выводим регистры FPU, также передаем указатель на структуру.

Для имитации lsmod восстанавливаем список модулей начиная с нашего собственного, дальнейшую информацию по имени вытаскиваем из /sys/module. На уровень пользователя передаем структуру содержащую указатели на наш

модуль, следующий модуль (структурами реализован связный список) и стандартный вывод lsmod.

Вывод программы:

```
Opening Driver
Writing Value to Driver
Reading Value from Driver
struct user_lsmod=0x7ffcc2b32370: {
  module_names=0xffffffffc096d1d8
  module_=0xffffffffc096d1c0
  next=0xfffffa86583067d98
  lsmod_len=2283
}
Module          Size      Used by
vboxsf          36864      0
binfmt_misc     24576      1
snd_intel8x0     45056      4
snd_ac97_codec   180224     1
ac97_bus         16384      1
snd_pcm          143360     3
snd_seq_midi     20480      0
snd_seq_midi_event 16384      1
nls_iso8859_1    16384      1
snd_rawmidi      49152      1
intel_rapl_msr   20480      0
intel_rapl_common 40960      1
crct10dif_pclmul 16384      1
ghash_clmulni_intel 16384      0
snd_seq          77824      2
aesni_intel     376832     0
crypto_simd      16384      1
snd_seq_device   16384      3
cryptd           24576      2
snd_timer        40960      3
rapl             20480      0
joydev           32768      0
snd              106496     13
input_leds       16384      0
serio_raw        20480      0
soundcore        16384      1
vboxguest        45056      3
mac_hid          16384      0
sch_fq_codel     20480      2
vmwgfx           368640     2
ttm              86016      1
drm_kms_helper   311296     1
```

```

cec                61440      1
rc_core            65536      1
fb_sys_fops        16384      1
syscopyarea        16384      1
sysfillrect        20480      1
sysimgblt          16384      1
ipmi_devintf       20480      0
ipmi_msghandler    122880     1
msr                16384      0
parport_pc         49152      0
ppdev              24576      0
lp                 28672      0
parport            69632      3
ramoops            32768      0
reed_solomon       28672      1
mtd                77824      0
pstore_blk         16384      0
pstore_zone        32768      1
efi_pstore         16384      0
drm                622592     5
ip_tables          32768      0
x_tables           53248      1
autofs4            49152      2
hid_generic        16384      0
crc32_pclmul       16384      0
psmouse           176128     0
ahci               45056      2
libahci            45056      1
usbhid             65536      0
i2c_piix4          32768      0
hid                151552     2
e1000              159744     0
pata_acpi          16384      0
video              61440      0
struct user_fpu_state=0x7ffcc2b32390: {
  control_word=37f
  state_word=0
  last_instr_opcode=0
  instr_pointer=0
  data_pointer=0
  ip_offset=0
  ip_selector=0
  operand_offset=0
  operand_selector=0
  fpu_state_=0xffff97e4f0479580
}
Closing Driver

```

Вывод:

При выполнении данной лабораторной работы была изучена работа с интерфейсом передачи (системным вызовом) `ioctl`, используемого для работы с драйверами устройств ввода/вывода, а также была переосмыслена вся прошедшая жизнь и каким образом мы сегодня оказались здесь. Были изучены и не побеждены нюансы передачи строк между пространством пользователя и пространством ядра.