ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки: Информатика и вычислительная техника

Дисциплина «Операционные системы»

Лабораторная работа №2

Вариант IOCTL: lsmod, 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 ярда, 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*)

#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"
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 len, loff_t *off)
{
        pr_info("Read Function\n");
        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 RD_LSMOD:
                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;
                //WARN_ON(0);
                pr_info("%-18s %-10s %-3s", "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;
                        //pr_info("Module name is %s\n",
mod->name);
                        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 {
                        //pr_info("open file");
                          //fs = get_fs();
                          //set_fs(get_ds());
                          //f->f_op->read(f, buff, 256,
&f->f_pos);
                          kernel_read(f1, buff1, 16, &f1->f_pos);
                          //set_fs(fs);
                          filp_close(f1, NULL);
                          for (ptr = buff1 + strlen(buff1) - 1;
(ptr >= buff1) && isspace(*ptr); --ptr);
                          ptr[1] = '\0';
                        }
                        //strcat(buff, "    ");
                        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 {
```

```
                        //pr_info("open file");
                         //fs = get_fs();
                         //set_fs(get_ds());
                         //f->f_op->read(f, buff, 256,
&f->f_pos);
                         kernel_read(f2, buff2, 16, &f2->f_pos);
                         //set_fs(fs);
                         filp_close(f2, NULL);
                         for (ptr = buff2 + strlen(buff2) - 1;
(ptr >= buff2) && isspace(*ptr); --ptr);
                         ptr[1] = '\0';
                        }
                        //pr_info("Module: %s\n", mod->name);
                        //pr_info("Size: %s\n", buff1);
                        //pr_info("Used by: %s\n", buff2);
                        pr_info("%-18s %-10s %-3s", 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);
                 if(copy_to_user((struct user_lsmod*) arg, &begin,
sizeof(begin)))
                            {
                                    pr_err("Data Write1 :
Err!\n");
                            }
                       //pr_info("Value = %d\n", value);
                       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");
        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("Lsmod & 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); }

int main()
{
        int fd;
        int32_t number;
        int32_t value;

        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));
     }
        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. По техническим причинам имитация lsmod выводится в dmesg, на уровень пользователя передаем структуру содержащую указатели на наш модуль, следующий модуль (структурами реализован связный список).

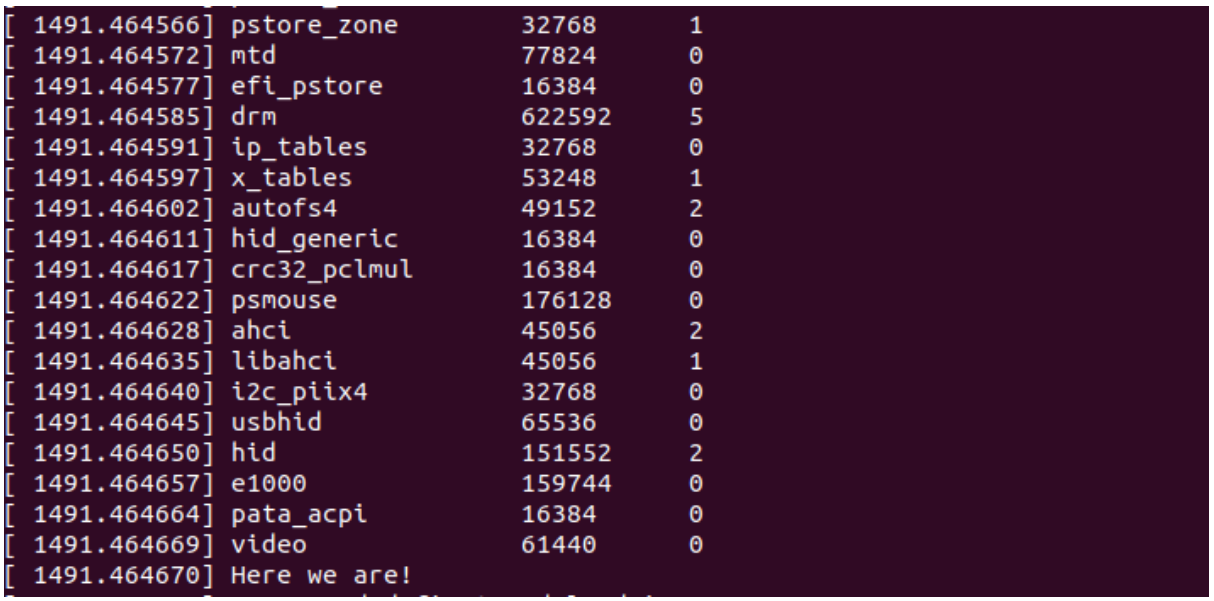
Вывод программы:

```
Opening Driver
Writing Value to Driver
Reading Value from Driver
For lsmod output please see dmsg
Pointer:
struct user_lsmod=0x7ffebd7d78b0: {
  module_names=0xffffffffc08c8198
  module_=0xffffffffc08c8180
  next=0xffffa60083173d70
}
struct user_fpu_state=0x7ffebd7d78d0: {
  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_=0xffff8edac8a04500
}
Closing Driver
```

dmesg:

```
[ 1491.464222] Device File Opened...!!!
[ 1491.464233] Module                  Size  Used by
[ 1491.464248] binfmt_misc            24576  1
[ 1491.464254] vboxsf                 36864  0
[ 1491.464260] snd_intel8x0           45056  2
[ 1491.464267] snd_ac97_codec        180224  1
[ 1491.464274] ac97_bus               16384  1
[ 1491.464280] snd_pcm                143360  2
[ 1491.464286] snd_seq_midi           20480  0
[ 1491.464294] snd_seq_midi_event     16384  1
[ 1491.464300] snd_rawmidi            49152  1
[ 1491.464307] intel_rapl_msr         20480  0
[ 1491.464313] nls_iso8859_1          16384  1
[ 1491.464320] intel_rapl_common      40960  1
[ 1491.464326] snd_seq                77824  2
[ 1491.464333] crct10dif_pclmul       16384  1
[ 1491.464339] ghash_clmulni_intel    16384   0
[ 1491.464345] snd_seq_device         16384  3
[ 1491.464351] snd_timer              40960  2
[ 1491.464357] aesni_intel           376832  0
[ 1491.464363] crypto_simd            16384  1
[ 1491.464369] cryptd                 24576  2
[ 1491.464375] rapl                   20480  0
[ 1491.464394] snd                   106496  11
```

```
[ 1491.464403] joydev                 32768  0
[ 1491.464410] input_leds             16384  0
[ 1491.464416] serio_raw              20480  0
[ 1491.464421] soundcore              16384  1
[ 1491.464427] vboxguest              45056  3
[ 1491.464434] mac_hid                16384  0
[ 1491.464440] sch_fq_codel           20480  2
[ 1491.464446] vmwgfx                368640  2
[ 1491.464453] ttm                    86016  1
[ 1491.464459] drm_kms_helper        311296  1
[ 1491.464466] cec                    61440  1
[ 1491.464471] rc_core                65536  1
[ 1491.464477] fb_sys_fops            16384  1
[ 1491.464484] syscopyarea            16384  1
[ 1491.464491] sysfillrect            20480  1
[ 1491.464496] sysimgblt              16384  1
[ 1491.464502] ipmi_devintf           20480  0
[ 1491.464511] ipmi_msghandler       122880  1
[ 1491.464518] msr                    16384  0
[ 1491.464524] parport_pc             49152  0
[ 1491.464528] ppdev                  24576  0
[ 1491.464535] lp                     28672  0
[ 1491.464542] parport                69632  3
[ 1491.464548] ramoops                32768  0
[ 1491.464553] reed_solomon           28672  1
[ 1491.464560] pstore_blk             16384  0
[ 1491.464566] pstore_zone            32768  1
```



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.