Федеральное государственное автономное образовательное учреждение высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  | |  |
|  |  | | |  |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 | | | | |
| Операционные системы | | | | |
| Вариант: ioctl: signal\_struct, page  Группа: P33121    Выполнил: Балтабаев Д. Т.  Преподаватель: Осипов С. В. | | | | |
|  |
|  |  | |
| Санкт-Петербург 2022 | | | | |

**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.

**Выполнение**

Изображение выглядит как текст

Автоматически созданное описание

**Модуль:**

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <linux/uaccess.h>

#include <linux/device.h>

#include <linux/vmalloc.h>

#include <linux/ioctl.h>

#include <linux/pid.h>

#include <linux/sched.h>

#include <linux/sched/signal.h>

#include <linux/seq\_file.h>

#include <asm/pgtable.h>

#include <asm/page.h>

#include <linux/pagemap.h>

#include <linux/mm\_types.h>

#include "linux/mm.h"

#define WR\_SIGNAL \_IOW('a','a',struct my\_signal\_struct\*)

#define WR\_PAGE \_IOW('a','b',struct my\_page\_struct\*)

#define RD\_SIGNAL \_IOR('b','b',struct my\_signal\_struct\*)

#define RD\_PAGE \_IOR('b','a',struct my\_page\_struct\*)

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_AUTHOR("Балтабаев Дамир P33121");

MODULE\_DESCRIPTION("ОСИ | Лабораторная работа №2 | ioctl: signal\_struct, page");

MODULE\_VERSION("1.0");

dev\_t first = 0; //Global variable for the first device number

static struct cdev c\_dev; //Global variable for the character device structure

static struct class \*cl; //Global variable for the device class

struct my\_signal\_struct {

bool valid;

int nr\_threads;

int group\_exit\_code;

int notify\_count;

int group\_stop\_count;

unsigned int flags;

pid\_t pid;

};

struct my\_page\_struct {

bool valid;

unsigned long flags;

long virtual\_address;

unsigned long mapping;

pid\_t pid;

};

struct task\_struct\* ts;

struct my\_signal\_struct mss;

struct my\_page\_struct mps;

static ssize\_t my\_ictl\_read(struct file \*file, char \_\_user \*buf, size\_t len,loff\_t \* off);

static ssize\_t my\_ictl\_write(struct file \*file, const char \*buf, size\_t len, loff\_t \* off);

static int my\_ictl\_open(struct inode \*inode, struct file \*file);

static long my\_ictl\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg);

static int my\_ictl\_release(struct inode \*inode, struct file \*file);

static int \_\_init ofcd\_init(void);

static void \_\_exit ofcd\_exit(void);

void signal\_struct\_method(void);

void page\_method(void);

static struct file\_operations fops =

{

.owner = THIS\_MODULE,

.read = my\_ictl\_read,

.write = my\_ictl\_write,

.open = my\_ictl\_open,

.unlocked\_ioctl = my\_ictl\_ioctl,

.release = my\_ictl\_release,

};

static int my\_ictl\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Была вызвана функция открытия!\n");

return 0;

}

static int my\_ictl\_release(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Была вызвана функция закрытия!\n");

return 0;

}

static ssize\_t my\_ictl\_read(struct file \*file, char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Была вызвана функция чтения!\n");

return 0;

}

static ssize\_t my\_ictl\_write(struct file \*file, const char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Была вызвана функция записи!\n");

return 0;

}

static long my\_ictl\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)

{

switch(cmd) {

case WR\_SIGNAL:

if( copy\_from\_user(&mss ,(struct my\_signal\_struct\*) arg, sizeof(mss)))

{

printk(KERN\_ERR "Ошибка копирования данных от пользователя!\n");

}

signal\_struct\_method();

break;

case WR\_PAGE:

if( copy\_from\_user(&mps ,(struct my\_page\_struct\*) arg, sizeof(mps)))

{

printk(KERN\_ERR "Ошибка копирования данных от пользователя!\n");

}

page\_method();

break;

case RD\_SIGNAL:

if( copy\_to\_user((struct my\_signal\_struct\*) arg, &mss , sizeof(mss)))

{

printk(KERN\_ERR "Ошибка копирования данных пользователю!\n");

}

break;

case RD\_PAGE:

if( copy\_to\_user((struct my\_page\_struct\*) arg, &mps , sizeof(mps)))

{

printk(KERN\_ERR "Ошибка копирования данных пользователю!\n");

}

break;

default:

printk(KERN\_INFO "Заданная команда нереализована!");

break;

}

return 0;

}

static int \_\_init ofcd\_init(void) /\* Constructor \*/

{

//регистрация MAJOR and MINOR

//int alloc\_chrdev\_region(dev\_t \*first, unsigned int firstminor, unsigned int cnt, char \*name);

if((alloc\_chrdev\_region(&first, 0, 1, "my\_new\_dev")) <0)

{

printk(KERN\_ERR "Не удалось зарегистрировать диапазон символьных номеров устройств\n");

return -1;

}

//создание класса устройства

if((cl = class\_create(THIS\_MODULE,"my\_new\_class")) == NULL)

{

printk(KERN\_ERR "Невозможно создать структуру класса\n");

unregister\_chrdev\_region(first,1);

return -1;

}

//заполнение класса информацией об устройстве (MAJOR,MINOR)

if((device\_create(cl,NULL,first,NULL,"my\_new\_dev")) == NULL)

{

printk(KERN\_ERR "Невозможно создать девайс\n");

class\_destroy(cl);

unregister\_chrdev\_region(first,1);

return -1;

}

//заполняем структуру устройства необходимыми функциями

cdev\_init(&c\_dev,&fops);

//добавляем девайс в систему

if((cdev\_add(&c\_dev,first,1)) == -1)

{

printk(KERN\_ERR "Невозможно добавить девайс в систему\n");

device\_destroy(cl,first);

class\_destroy(cl);

unregister\_chrdev\_region(first, 1);

return -1;

}

printk(KERN\_INFO "Установка драйвера произошла успешно!\n");

return 0;

}

void signal\_struct\_method(){

ts = get\_pid\_task(find\_get\_pid(mss.pid), PIDTYPE\_PID);

if (ts == NULL){

mss.valid = false;

return 0;

}else{

struct signal\_struct \*signalStruct = ts->signal;

mss.valid = true;

mss.nr\_threads = signalStruct->nr\_threads;

mss.group\_exit\_code = signalStruct->group\_exit\_code;

mss.notify\_count = signalStruct->notify\_count;

mss.group\_stop\_count = signalStruct->group\_stop\_count;

mss.flags = signalStruct->flags;

}

}

static struct page \*get\_current\_page(struct mm\_struct\* mm, long virtual\_address) {

pgd\_t \*pgd;

p4d\_t \*p4d;

pud\_t \*pud;

pmd\_t \*pmd;

pte\_t \*pte;

struct page \*page = NULL;

pgd = pgd\_offset(mm, virtual\_address);

if (!pgd\_present(\*pgd)) {

return NULL;

}

p4d = p4d\_offset(pgd, virtual\_address);

if (!p4d\_present(\*p4d)) {

return NULL;

}

pud = pud\_offset(p4d, virtual\_address);

if (!pud\_present(\*pud)) {

return NULL;

}

pmd = pmd\_offset(pud, virtual\_address);

if (!pmd\_present(\*pmd)) {

return NULL;

}

pte = pte\_offset\_kernel(pmd, virtual\_address);

if (!pte\_present(\*pte)) {

return NULL;

}

page = pte\_page(\*pte);

return page;

}

void page\_method(){

ts = get\_pid\_task(find\_get\_pid(mps.pid), PIDTYPE\_PID);

if (ts == NULL){

mps.valid = false;

return 0;

}else {

struct page \*page\_struct;

struct mm\_struct \*mm = ts->mm;

if (mm == NULL)

{

mps.valid=false;

return 0;

}else {

mps.valid=true;

struct vm\_area\_struct \*vas = mm->mmap;

long virtual\_address;

for (virtual\_address = vas->vm\_start; virtual\_address <= vas->vm\_end;virtual\_address += PAGE\_SIZE){

page\_struct = get\_current\_page(mm, virtual\_address);

if (page\_struct != NULL) {

mps.flags = page\_struct->flags;

mps.virtual\_address = virtual\_address;

mps.mapping= page\_struct->mapping;

break;

}

}

if (page\_struct == NULL)

{

mps.valid=false;

printk(KERN\_ERR "Ошибка при взятии страницы!\n");

return 0;

}

}

}

}

static void \_\_exit ofcd\_exit(void) /\*Desctructor \*/

{

cdev\_del(&c\_dev);

device\_destroy(cl, first);

class\_destroy(cl);

unregister\_chrdev\_region(first, 1);

printk(KERN\_INFO "Девайс уничтожен!");

}

module\_init(ofcd\_init);

module\_exit(ofcd\_exit);

**Пользовательская программа:**

#include <sys/ioctl.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <malloc.h>

#include <unistd.h>

#include <stdbool.h>

#define WR\_SIGNAL \_IOW('a','a',struct my\_signal\_struct\*)

#define WR\_PAGE \_IOW('a','b',struct my\_page\_struct\*)

#define RD\_SIGNAL \_IOR('b','b',struct my\_signal\_struct\*)

#define RD\_PAGE \_IOR('b','a',struct my\_page\_struct\*)

struct my\_signal\_struct {

bool valid;

int nr\_threads;

int group\_exit\_code;

int notify\_count;

int group\_stop\_count;

unsigned int flags;

pid\_t pid;

};

struct my\_page\_struct {

bool valid;

unsigned long flags;

long virtual\_address;

unsigned long mapping;

pid\_t pid;

};

int main(int argc, char \*argv[]) {

if (argc < 3){

printf("Вам необходимо ввести struct\_id (1 - signal, 2 - page) и PID!\n" );

return 0;

}

if (argc > 3){

printf("Некорректное количество аргументов!\n" );

return 0;

}

int struct\_id = atoi(argv[1]);

pid\_t console\_pid = atoi(argv[2]);

int dev;

if (struct\_id == 1){

struct my\_signal\_struct mss;

mss.pid = console\_pid;

printf("\nОткрытие драйвера...");

dev = open("/dev/my\_new\_dev", O\_RDWR);

if (dev == -1){

printf("\nОшибка при открытии!\n");

return -1;

}

printf("\nОткрытие прошло успешно!\n");

printf("Запись данных в драйвер...\n");

ioctl(dev, WR\_SIGNAL, (struct my\_signal\_struct\*) &mss);

printf("Чтение данных из драйвера...\n");

ioctl(dev, RD\_SIGNAL, (struct my\_signal\_struct\*) &mss);

if (mss.valid == true){

printf("\nSignal struct, PID %d: \n", mss.pid);

printf("\tnr\_threads = %d\n", mss.nr\_threads);

printf("\tgroup\_exit\_code = %d\n", mss.group\_exit\_code);

printf("\tnotify\_count = %d\n", mss.notify\_count);

printf("\tgroup\_stop\_count = %d\n", mss.group\_stop\_count);

printf("\tflags = %u\n", mss.flags);

}else printf("\nОшибка! Заданный PID не существует!");

printf("\nЗакрытие драйвера...\n");

close(dev);

}

else if (struct\_id == 2){

struct my\_page\_struct mps;

mps.pid = console\_pid;

printf("\nОткрытие драйвера...");

dev = open("/dev/my\_new\_dev", O\_RDWR);

if (dev == -1){

printf("\nОшибка при открытии!\n");

return -1;

}

printf("\nОткрытие прошло успешно!\n");

printf("Запись данных в драйвер...\n");

ioctl(dev, WR\_PAGE, (struct my\_page\_struct\*) &mps);

printf("Чтение данных из драйвера...\n");

ioctl(dev, RD\_PAGE, (struct my\_page\_struct\*) &mps);

if (mps.valid == true){

printf("\nPage struct, PID %d: \n", mps.pid);

printf("\tflags = %ld\n", mps.flags);

printf("\tvirtual address = %lx\n", mps.virtual\_address);

printf("\tphysical address = %lx\n", mps.mapping);

}else printf("\nОшибка! Невозможно получить структуру page!");

printf("\nЗакрытие драйвера...\n");

close(dev);

} else {

printf("\nОшибка! Введенный struct\_id не существует!\n");

return 0;

}

}

**Makefile:**

obj-m += my\_module.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

**Вывод**

В ходе работы я познакомился с понятием интерфейса ioctl, реализовав программу на пользовательском уровне, выводящую необходимую информацию в удобном для чтения человеком виде и программу на уровне ядра, собирающую необходимую информацию. Также поработал с такими структурами как: struct\_signal и page.