МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине

“Операционные системы”

Вариант: ioctl: page, syscall\_info

**Студент:**

Воробьев К.О.

            Группа P33302

**Преподаватель:**

Осипов С. В.



Санкт-Петербург, 2022

**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.

Изображение выглядит как текст, человек, внешний

Автоматически созданное описание

**The struct page is defined in include/linux/mm\_types.h as:**

Изображение выглядит как текст

Автоматически созданное описание

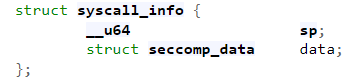
Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

**The struct syscall\_info is defined in include/linux/ptrace.h as:**



**Выполнение:**

**/cmd\_mode.h**

#define WR\_PID \_IOW('a','a', int32\_t\*)  
#define RD\_MY\_PAGE \_IOR('a','c', struct my\_page\*)  
#define RD\_MY\_SYSCALL\_INFO \_IOR('a','d', struct my\_syscall\_info\*)  
  
struct my\_seccomp\_data {  
 int nr;  
 \_\_u32 arch;  
 \_\_u64 instruction\_pointer;  
 \_\_u64 args[6];  
};  
  
struct my\_syscall\_info {  
 \_\_u64 sp;  
 struct my\_seccomp\_data data;  
};  
  
struct my\_page {  
 unsigned long flags;  
 void\* mapping;  
 long vm\_address;  
};

**/core\_mod.c**

#include <linux/init.h>  
#include <linux/module.h>  
#include <linux/kernel.h>  
#include <linux/ioctl.h>  
#include <linux/ptrace.h>  
#include <linux/pid.h>  
#include <linux/netdevice.h>  
#include <asm/syscall.h>  
#include <linux/sched.h>  
#include <linux/namei.h>  
#include <linux/mm\_types.h>  
#include <asm/page.h>  
  
#include "cmd\_mode.h"  
  
MODULE\_LICENSE("GPL");  
MODULE\_AUTHOR("Kirill Vorobyev");  
MODULE\_DESCRIPTION("My ioctl driver");  
MODULE\_VERSION("1.0");  
  
const int MYMAJOR = 28;  
  
int pid = 0;  
  
static void print\_my\_page(const struct my\_page\* mp) {  
 printk("\nPage:\n");  
 printk("flags: %lu\n", mp->flags);  
 printk("Virtual address: %ld, Page address: %p\n", mp->vm\_address, mp->mapping);  
}  
  
static void print\_my\_syscall\_info(const struct my\_syscall\_info\* msi) {  
 printk("\nSyscall\_info:\n");  
 printk("sp %lld\n", msi->sp);  
 printk("nr %d\n", msi->data.nr);  
 printk("instruction\_pointer %lld\n", msi->data.instruction\_pointer);  
 printk("arg %d 0x%08X\n", 0, (unsigned int) msi->data.args[0]);  
 printk("arg %d 0x%08X\n", 1, (unsigned int) msi->data.args[1]);  
 printk("arg %d 0x%08X\n", 2, (unsigned int) msi->data.args[2]);  
 printk("arg %d 0x%08X\n", 3, (unsigned int) msi->data.args[3]);  
 printk("arg %d 0x%08X\n", 4, (unsigned int) msi->data.args[4]);  
 printk("arg %d 0x%08X\n", 5, (unsigned int) msi->data.args[5]);  
}  
  
static struct page\* get\_page\_by\_mm\_and\_address(struct mm\_struct\* mm, long address) {  
 pgd\_t\* pgd;  
 p4d\_t\* p4d;  
 pud\_t\* pud;  
 pmd\_t\* pmd;  
 pte\_t\* pte;  
 struct page\* page;  
 pgd = pgd\_offset(mm, address);  
 if (!pgd\_present(\*pgd)) return NULL;  
 p4d = p4d\_offset(pgd, address);  
 if (!p4d\_present(\*p4d)) return NULL;  
 pud = pud\_offset(p4d, address);  
 if (!pud\_present(\*pud)) return NULL;  
 pmd = pmd\_offset(pud, address);  
 if (!pmd\_present(\*pmd)) return NULL;  
 pte = pte\_offset\_map(pmd, address);  
 if (!pte\_present(\*pte)) return NULL;  
 page = pte\_page(\*pte);  
 return page;  
}  
  
// 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) {  
 struct my\_page mp;  
 struct my\_syscall\_info msi;  
 struct page\* page;  
 struct task\_struct\* task;  
 struct mm\_struct\* mm;  
 struct vm\_area\_struct\* vm;  
 switch(cmd) {  
 case WR\_PID:  
 if(copy\_from\_user(&pid, (int\*) arg, sizeof(pid))) pr\_err("Data write error!\n");  
 pr\_info("Pid = %d\n", pid);  
 break;  
 case RD\_MY\_PAGE:  
 task = get\_pid\_task(find\_get\_pid(pid), PIDTYPE\_PID);  
 mm = task->mm;  
 vm = mm->mmap;  
 if (mm == NULL) {  
 printk(KERN\_INFO "Task struct hasn't mm\n");  
 }  
 long address = vm->vm\_start;  
 long end = vm->vm\_end;  
 while (address <= end) {  
 page = get\_page\_by\_mm\_and\_address(mm, address);  
 if (page != NULL) {  
 mp.flags = page->flags;  
 mp.mapping = (void\*)page->mapping;  
 mp.vm\_address = address;  
 break;  
 }  
 address += PAGE\_SIZE;  
 }  
 print\_my\_page(&mp);  
 if(copy\_to\_user((struct my\_page\*) arg, &mp, sizeof(struct my\_page))) {  
 printk(KERN\_INFO "Data read error!\n");  
 }  
 break;  
 case RD\_MY\_SYSCALL\_INFO:  
 task = get\_pid\_task(find\_get\_pid(pid), PIDTYPE\_PID);  
 unsigned long args[6] = {};  
 struct pt\_regs\* regs = task\_pt\_regs(task);  
 msi.sp = user\_stack\_pointer(regs);  
 msi.data.instruction\_pointer = instruction\_pointer(regs);  
 msi.data.nr = syscall\_get\_nr(task, regs);  
 if (msi.data.nr != -1L) syscall\_get\_arguments(task, regs, args);  
 print\_my\_syscall\_info(&msi);  
 if(copy\_to\_user((struct my\_syscall\_info\*) arg, &msi, sizeof(struct my\_syscall\_info))) printk(KERN\_INFO "Data read error!\n");  
 break;  
 default:  
 pr\_info("Command not found!");  
 break;  
 }  
 return 0;  
}  
  
static struct file\_operations fops = {  
 .owner = THIS\_MODULE,  
 .unlocked\_ioctl = driver\_ioctl,  
};  
  
static int \_\_init ioctl\_core\_init(void) {  
 printk(KERN\_INFO "Core mode is started, hello, world!\n");  
 int retval;  
 retval = register\_chrdev(MYMAJOR, "my\_driver", &fops);  
 if (0 == retval) {  
 printk("my\_driver device number Major:%d , Minor:%d\n", MYMAJOR, 0);  
 } else if (retval > 0) {  
 printk("my\_driver device number Major:%d , Minor:%d\n", retval >> 20, retval & 0xffff);  
 } else {  
 printk("Couldn't register device number!\n");  
 return -1;  
 }  
 return 0;  
}  
  
static void \_\_exit ioctl\_core\_exit(void) {  
 unregister\_chrdev(MYMAJOR, "my\_driver");  
 printk(KERN\_INFO "Core mode is finished, goodbye!\n");  
}  
  
module\_init(ioctl\_core\_init);  
module\_exit(ioctl\_core\_exit);

**/user\_mode.c**

#include <sys/ioctl.h>  
#include <stdio.h>  
#include <stdint.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <malloc.h>  
#include <unistd.h>  
#include <linux/ptrace.h>  
  
#include "cmd\_mode.h"  
  
static void print\_line(void) {  
 printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  
}  
  
static void print\_my\_syscall\_info(const struct my\_syscall\_info\* si) {  
 printf("\nSyscall\_info:\n\n");  
 printf("sp %lld\n", si->sp);  
 printf("nr %d\n", si->data.nr);  
 printf("instruction\_pointer %lld\n", si->data.instruction\_pointer);  
 for (int i = 0; i < 6; i++) printf("arg %d 0x%08X\n", i, (unsigned int) si->data.args[i]);  
}  
  
static void print\_page(const struct my\_page\* mp) {  
 printf("\nPage:\n\n");  
 printf("flags: %ld\n", mp->flags);  
 printf("Virtual address: %ld\n", mp->vm\_address);  
 printf("Page address: %p\n", mp->mapping);  
}  
  
int main(int argc, char \*argv[]) {  
 int fd;  
 if(argc < 2) {  
 printf("The program needs an argument - a pid!\n");  
 return -1;  
 }  
 int32\_t pid = atoi(argv[1]);  
 if (pid < 1) {  
 printf("Pid can be only greater than 0\n");  
 return -1;  
 }  
 printf("\nOpening a driver...\n");  
 fd = open("/dev/etc\_device", O\_WRONLY);  
 if(fd < 0) {  
 printf("Cannot open device file:(\n");  
 return 0;  
 }  
 struct my\_page mp;  
 struct my\_syscall\_info msi;  
 // Writing a pid to driver  
 ioctl(fd, WR\_PID, (int32\_t\*) &pid);  
 // Reading my\_page from driver  
 ioctl(fd, RD\_MY\_PAGE, &mp);  
 print\_line();  
 print\_page(&mp);  
 // Reading my\_syscall\_info from driver  
 ioctl(fd, RD\_MY\_SYSCALL\_INFO, &msi);  
 print\_line();  
 print\_my\_syscall\_info(&msi);  
 print\_line();  
 printf("Closing a driver...\n");  
 close(fd);  
}

**Изображение выглядит как текст

Автоматически созданное описание**

**Вывод:**

Во время выполнения лабораторной работы я углубился в работу ядра linux. Написал собственный модуль ядра и клиентское приложение, для работы с этим модулем, реализовав общение между ними с помощью ioctl.