Университет ИТМО
Факультет ПИиКТ

Операционные системы
Лабораторная работа №2

Работу выполнил:
Конаныхина А.А.

Группа:
P33102

Вариант:
ioctl: thread_struct, net_device

Преподаватель:
Барсуков И.А.

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.

Выполнение:

module_lab2.c:

```
#include <linux/init.h>

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

```c
#include <asm/processor.h>
#include <linux/netdevice.h>
#include <linux/pid.h>
#include <linux/sched.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Linux module for OS Lab2");
MODULE_VERSION("1.0");

struct thread_msg{
unsigned long sp;

unsigned short es;
unsigned short ds;
unsigned short fsindex;
unsigned short gsindex;

unsigned long fsbase;
unsigned long gsbase;

unsigned long virtual_dr6;

unsigned long ptrace_dr7;

unsigned long cr2;
unsigned long trap_nr;
unsigned long error_code;

unsigned int pkru;
};

struct net_msg{
unsigned int flags;
unsigned long mem_end;
unsigned long mem_start;
unsigned long base_addr;
int irq;
unsigned long state;
unsigned char if_port;
unsigned char dma;
};

struct message {
struct thread_msg th;
struct net_msg nt;
pid_t pid;
};

#define WR_VALUE _IOWR('a','a',struct message*)

dev_t dev = 0;
static struct class *dev_class;
static struct cdev etx_cdev;

struct thread_struct* thread;
struct net_device* n_dev;
struct message msg;

static int etx_open(struct inode *inode, struct file *file);
static int etx_release(struct inode *inode, struct file *file);
```

```
static ssize_t etx_read(struct file *filp, char __user *buf, size_t len,loff_t * off);
static ssize_t etx_write(struct file *filp, const char *buf, size_t len, loff_t * off);
static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg);

static int etx_open(struct inode *inode, struct file *file) {
pr_info("Device File Opened...!!!\n");
return 0;
}



static int etx_release(struct inode *inode, struct file *file) {
pr_info("Device File Closed...!!!\n");
return 0;
}

static ssize_t etx_read(struct file *filp, char __user *buf, size_t len, loff_t *off) {
pr_info("Read Function\n");
return 0;
}

static ssize_t etx_write(struct file *filp, const char __user *buf, size_t len, loff_t *off) {
pr_info("Write function\n");
return len;
}

void read_thread_struct(void)
{
struct task_struct* ts = get_pid_task(find_get_pid(msg.pid), PIDTYPE_PID);
if (ts)
{
thread = &(ts->thread);
if (thread)
{
msg.th.sp = thread->sp;

msg.th.es = thread->es;
msg.th.ds = thread->ds;
msg.th.fsindex = thread->fsindex;
msg.th.gsindex = thread->gsindex;

msg.th.fsbase = thread->fsbase;
msg.th.gsbase = thread->gsbase;

msg.th.virtual_dr6 = thread->virtual_dr6;

msg.th.ptrace_dr7 = thread->ptrace_dr7;

msg.th.cr2 = thread->cr2;
msg.th.trap_nr = thread->trap_nr;
msg.th.error_code = thread->error_code;

msg.th.pkru = thread->pkru;
}
}
}

void read_net_device(void)
{
struct net* net = get_net_ns_by_pid(msg.pid);
if (net)
{
```

```
n_dev = first_net_device(net);
if (n_dev)
{
msg.nt.flags = n_dev->flags;
msg.nt.mem_end = n_dev->mem_end;
msg.nt.mem_start = n_dev->mem_start;
msg.nt.base_addr = n_dev->base_addr;
msg.nt.irq = n_dev->irq;
msg.nt.dma = n_dev->dma;
msg.nt.if_port = n_dev->if_port;
msg.nt.state = n_dev->state;
}
}
}

static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg) {
switch(cmd) {
case WR_VALUE:
pr_info("WR_VALUE\n");
if(copy_from_user(&msg, (struct message*) arg, sizeof(msg)))
{
pr_err("Data Read: Err!\n");
}
pr_info("ID: %d\n", msg.pid);

read_net_device();
read_thread_struct();

pr_info("Return value\n");
if (copy_to_user((struct message*) arg, &msg, sizeof(msg)))
{
pr_err("Data Write: Err!\n");
}
break;
default:
pr_info("Default\n");
break;
}
return 0;
}

static struct file_operations fops =
{
.owner = THIS_MODULE,
.read = etx_read,
.write = etx_write,
.open = etx_open,
.unlocked_ioctl = etx_ioctl,
.release = etx_release,
};

static int __init module_start(void)
{
/*Allocating Major number*/
if((alloc_chrdev_region(&dev, 0, 1, "etx_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(&etx_cdev,&fops);
```

```
/*Adding character device to the system*/
if((cdev_add(&etx_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,"etx_class"))){
pr_err("Cannot create the struct class\n");
goto r_class;
}

/*Creating device*/
if(IS_ERR(device_create(dev_class,NULL,dev,NULL,"etx_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;
}

static void __exit module_end(void)
{
device_destroy(dev_class,dev);
class_destroy(dev_class);
cdev_del(&etx_cdev);
unregister_chrdev_region(dev, 1);
pr_info("Device Driver Remove...Done!!!\n");
}

module_init(module_start);
module_exit(module_end);
```

user.c:

```
#include <sys/ioctl.h>

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/poll.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


struct thread_msg{
```

```
unsigned long sp;

unsigned short es;
unsigned short ds;
unsigned short fsindex;
unsigned short gsindex;

unsigned long fsbase;
unsigned long gsbase;

unsigned long virtual_dr6;

unsigned long ptrace_dr7;

unsigned long cr2;
unsigned long trap_nr;
unsigned long error_code;

unsigned int pkru;
};

struct net_msg{
unsigned int flags;
unsigned long mem_end;
unsigned long mem_start;
unsigned long base_addr;
int irq;
unsigned long state;
unsigned char if_port;
unsigned char dma;
};

struct message {
struct thread_msg th;
struct net_msg nt;
pid_t pid;
};

#define WR_VALUE _IOWR('a','a',struct message*)

int main(int argc, char *argv[]) {
struct message msg;
int fde;

msg.pid = atoi(argv[1]);

printf("\nOpening Driver\n");
printf("%d\n", msg.pid);
fde = open("/dev/etx_device", O_RDWR);

if(fde < 0) {
printf("Cannot open device file...\n");
printf("%d\n", errno);
return 0;
}

printf("Writing data to Driver\n");
ioctl(fde, WR_VALUE, (struct message*) &msg);

printf ("Net flags: %ld\n", msg.nt.flags);
printf ("Net mem end: %ld\n", msg.nt.mem_end);
```

```
printf ("Net mem start: %ld\n", msg.nt.mem_start);
printf ("Net base addr: %ld\n", msg.nt.base_addr);
printf ("Net irq: %ld\n", msg.nt.irq);
printf ("Net dma: %ld\n", msg.nt.dma);
printf ("Net flags: %ld\n", msg.nt.flags);
printf ("Net port (if exist): %ld\n", msg.nt.if_port);
printf ("Net state: %ld\n", msg.nt.state);
printf ("\n");

printf ("Thread cr2: %ld\n", msg.th.cr2);
printf ("Thread ds: %ld\n", msg.th.ds);
printf ("Thread error_code: %ld\n", msg.th.error_code);
printf ("Thread es: %ld\n", msg.th.es);
printf ("Thread fsbase: %ld\n", msg.th.fsbase);
printf ("Thread fsindex: %ld\n", msg.th.fsindex);
printf ("Thread gsbase: %ld\n", msg.th.gsbase);
printf ("Thread gsindex: %ld\n", msg.th.gsindex);
printf ("Thread pkru: %ld\n", msg.th.pkru);
printf ("Thread ptrace_dr7: %ld\n", msg.th.ptrace_dr7);
printf ("Thread sp: %d\n", msg.th.sp);
printf ("Thread trap_nr: %ld\n", msg.th.trap_nr);
printf ("Thread cirtual_dr6: %ld\n", msg.th.virtual_dr6);

printf("\nClosing Driver\n");
close(fde);
}
```

Вывод:

В ходе выполнения данной лабораторной работы был глубже изучен исходный код Linux и написан собственный модуль ядра. Грустно и невкусно, но очень интересно.