Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Операционные системы

Лабораторная работа 2

Выполнил студент:
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Осипов Святослав Владимирович

Санкт-Петербург

2021 г.

Текст задания

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:

Procfs - файловая система /proc, передача параметров через запись в файл.

Целевая структура:

Имя структуры задается в заголовочных файлах Linux

Выполнение

Структура 1: net_device

Для поиска структур я пользовался сайтом https://elixir.bootlin.com . Данная структура определена в файле /include/linux/netdevice.h
Для того, чтобы понять, как получить данную структуру, я также воспользовался поиском на сайте.

Данную структуру можно получить через фукнцию **first_net_device**:

```
static inline struct net_device *first_net_device(struct net *net)
{
    return list_empty(&net->dev_base_head) ? NULL :
        net_device_entry(net->dev_base_head.next);
}
```

Для получения остальных девайсов мы можем в цикле использовать функцию **next_net_device**:

```
static inline struct net_device *next_net_device(struct net_device *dev)
{
    struct list_head *lh;
    struct net *net;

    net = dev_net(dev);
    lh = dev->dev_list.next;
    return lh == &net->dev_base_head ? NULL : net_device_entry(lh);
}
```

Также нужно использовать блокировку, что список не обновлялся.

```
read_lock(&dev_base_lock);
...
read_unlock(&dev_base_lock);
```

Структура 2: signal_struct

Данная структура определена в файле /include/linux/sched/signal.h
Для того, чтобы понять, как получить данную структуру, я также воспользовался поиском на сайте.
Чтобы получить указатель на данную структуру мы можем воспользоваться task_struct (пример был найден в /kenel/taskstats.c):

```
struct signal_struct *sig = task_struct->signal;
```

Получить же указатель на структуру task_struct можно с помощью следующих функций (пример был найден в /include/linux/pid.h):

```
get_pid_task(find_get_pid(pid), PIDTYPE_PID);
```

Таким образом, для получения указанной структуры, необходимо по номеру PID получить task_struct, далее из данной структуры уже получить нашу искомую структуру signal_sturct.

Код модуля ядра

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>                 // kmalloc()
#include <linux/uaccess.h>              // copy_to/from_user()
#include <linux/proc_fs.h>

#include <linux/pid.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>
#include <linux/netdevice.h>
#include <linux/device.h>

#define BUF_SIZE 1024

MODULE_LICENSE("Dual BSD/GPL");
MODULE_DESCRIPTION("Stab linux module for operating system's lab");
MODULE_VERSION("1.0");

static int pid = 1;
static int struct_id = 1;

static struct proc_dir_entry *parent;

/*
** Function Prototypes
*/
static int      __init lab_driver_init(void);
static void     __exit lab_driver_exit(void);


/***************** Procfs Functions *******************/
static int      open_proc(struct inode *inode, struct file *file);
static int      release_proc(struct inode *inode, struct file *file);
static ssize_t  read_proc(struct file *filp, char __user *buffer, size_t
length,loff_t * offset);
static ssize_t  write_proc(struct file *filp, const char *buff, size_t len,
loff_t * off);

/*
** procfs operation sturcture
*/
static struct proc_ops proc_fops = {
        .proc_open = open_proc,
        .proc_read = read_proc,
        .proc_write = write_proc,
```

```
        .proc_release = release_proc
};

// net_device
static size_t write_net_device_struct(char __user *ubuf){
    char buf[BUF_SIZE];
    size_t len = 0;

    static struct net_device *dev;

    read_lock(&dev_base_lock);

    dev = first_net_device(&init_net);
    while(dev){
        len += sprintf(buf+len, "found [%s]\n",     dev->name);
        len += sprintf(buf+len, "base_addr = %ld\n", dev->base_addr);
        len += sprintf(buf+len, "mem_start = %ld\n", dev->mem_start);
        len += sprintf(buf+len, "mem_end = %ld\n\n", dev->mem_end);
        dev = next_net_device(dev);
    }

    read_unlock(&dev_base_lock);

    if (copy_to_user(ubuf, buf, len)){
        return -EFAULT;
    }

    return len;
}

// signal_struct
static size_t write_signal_struct(char __user *ubuf, struct task_struct
*task_struct_ref){
    char buf[BUF_SIZE];
    size_t len = 0;

    struct signal_struct *signalStruct = task_struct_ref->signal;

    len += sprintf(buf,     "live = %d\n",
atomic_read(&(signalStruct->live)));
    len += sprintf(buf+len, "nr_threads = %d\n",            signalStruct-
>nr_threads);
    len += sprintf(buf+len, "group_exit_code = %d\n",       signalStruct-
>group_exit_code);
    len += sprintf(buf+len, "notify_count = %d\n",          signalStruct-
>notify_count);
    len += sprintf(buf+len, "group_stop_count = %d\n",      signalStruct-
>group_stop_count);
    len += sprintf(buf+len, "flags = %d\n",                 signalStruct-
>flags);
    len += sprintf(buf+len, "is_child_subreaper = %d\n",    signalStruct-
>is_child_subreaper);
    len += sprintf(buf+len, "has_child_subreaper = %d\n",   signalStruct-
>has_child_subreaper);

    if (copy_to_user(ubuf, buf, len)){
        return -EFAULT;
    }

    return len;
}


/*
```

```
** Эта фануция будет вызвана, когда мы ОТКРОЕМ файл procfs
*/
static int open_proc(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "proc file opend.....\t");
    return 0;
}

/*
** Эта фануция будет вызвана, когда мы ЗАКРОЕМ файл procfs
*/
static int release_proc(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "proc file released.....\n");
    return 0;
}

/*
** Эта фануция будет вызвана, когда мы ПРОЧИТАЕМ файл procfs
*/
static ssize_t read_proc(struct file *filp, char __user *ubuf, size_t count,
loff_t *ppos) {

    char buf[BUF_SIZE];
    int len = 0;
    struct task_struct *task_struct_ref = get_pid_task(find_get_pid(pid),
PIDTYPE_PID);

    printk(KERN_INFO "proc file read.....\n");
    if (*ppos > 0 || count < BUF_SIZE){
        return 0;
    }

    if (task_struct_ref == NULL){
        len += sprintf(buf,"task_struct for pid %d is NULL. Can not get any
information\n",pid);

        if (copy_to_user(ubuf, buf, len)){
            return -EFAULT;
        }
        *ppos = len;
        return len;
    }

    switch(struct_id){
        default:
        case 0:
            len = write_net_device_struct(ubuf);
            break;
        case 1:
            len = write_signal_struct(ubuf, task_struct_ref);
            break;
    }

    *ppos = len;
    return len;
}

/*
** Эта фануция будет вызвана, когда мы ЗАПИШЕМ в файл procfs
*/
static ssize_t write_proc(struct file *filp, const char __user *ubuf, size_t
count, loff_t *ppos) {
```

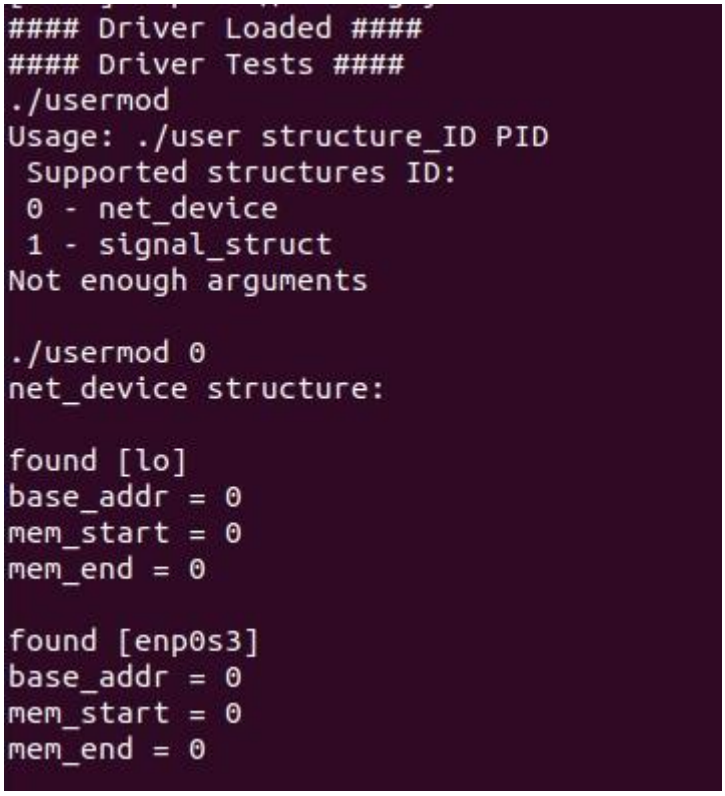
```
    int num_of_read_digits, c, a, b;
    char buf[BUF_SIZE];

    printk(KERN_INFO "proc file wrote.....\n");

    if (*ppos > 0 || count > BUF_SIZE){
        return -EFAULT;
    }

    if( copy_from_user(buf, ubuf, count) ) {
        return -EFAULT;
    }

    num_of_read_digits = sscanf(buf, "%d %d", &a, &b);
    if (num_of_read_digits != 2){
        return -EFAULT;
    }

    struct_id = a;
    pid = b;

    c = strlen(buf);
    *ppos = c;
    return c;
}

/*
** Функция инициализации Модуля
*/
static int __init lab_driver_init(void) {
    /* Создание директории процесса. Она будет создана в файловой системе
"/proc" */
    parent = proc_mkdir("lab",NULL);

    if( parent == NULL )
    {
        pr_info("Error creating proc entry");
        return -1;
    }

    /* Создание записи процесса в разделе "/proc/lab/" */
    proc_create("struct_info", 0666, parent, &proc_fops);

    pr_info("Device Driver Insert...Done!!!\n");
    return 0;
}

/*
** Функция выхода из Модуля
*/
static void __exit lab_driver_exit(void)
{
    /* Удаляет 1 запись процесса */
    //remove_proc_entry("lab/struct_info", parent);

    /* Удяление полностью /proc/lab */
    proc_remove(parent);

    pr_info("Device Driver Remove...Done!!!\n");
}

module_init(lab_driver_init);
module_exit(lab_driver_exit);
```

Код пользовательского приложения

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

void help(){
    fprintf(stderr, "Usage: ./user structure_ID PID\n "
                    "Supported structures ID: \n "
                    "0 - net_device\n "
                    "1 - signal_struct\n");
}

int main(int argc, char *argv[]){
    if (argc != 3) help();
    if (argc < 3){
        fprintf(stderr, "Not enough arguments \n" );
        return 0;
    }
    if (argc > 3){
        fprintf(stderr, "Too many arguments \n" );
        return 0;
    }

    char *p;
    errno = 0;
    long structure_ID = strtol(argv[1], &p, 10);
    if (*p != '\0' || errno != 0){
        fprintf(stderr, "Provided structure_ID must be number.\n");
        help();
        return 0;
    }
    errno = 0;
    long PID = strtol(argv[2], &p, 10);
    if (*p != '\0' || errno != 0){
        fprintf(stderr, "Provided PID must be number.\n");
        help();
        return 0;
    }

    if (structure_ID !=0 && structure_ID !=1){
        fprintf(stderr, "Provided structure ID is not supported.\n");
        help();
        return 0;
    }

    if(PID<0){
        fprintf(stderr, "PID must be positive \n" );
        help();
        return 0;
    }

    char inbuf[4096];
    char outbuf[4096];
    int fd = open("/proc/lab/struct_info", O_RDWR);
    sprintf(inbuf, "%s %s", argv[1], argv[2]);

    write(fd, inbuf, 17);
```

```
    lseek(fd, 0, SEEK_SET);
    read(fd, outbuf, 4096);

    if (structure_ID == 0){
        printf("net_device structure: \n\n");
    } else {
        printf("signal_struct structure data for PID %ld: \n\n", PID);
    }
    puts(outbuf);
    return 0;
}
```

Пример работы



Вывод

В ходе выполнения лабораторной работы я разработал комплекс программ на пользовательском уровне и уровне ядра, который собирают информацию на стороне ядра и передает информацию на уровень пользователя и выводит её в удобном для чтения человеком виде. В качестве интерфейса взаимодействия была использована файловая система /proc.

Программа на уровне пользователя принимает на вход аргументы командной строки, валидирует их, и, если они валидны, записывает их в файл /proc/lab/struct_info. Далее программа читает из этого файла уже саму информацию о запрошенной структуре.

Модуль ядра при записи в файл снимает с него информацию о требуемой структуре и PID (Если нужно). При чтении файла модуль ядра находит требуемую структуру и выводит информацию о ней в файл.

К сожалению, структуру bpf_sock_ops_kern так и не получилось достать по исходному варианту в текущей версии ядра 5.11. Поэтому согласовав с преподавателем я заменил её на структуру net_device.