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”

выполнил:

Иванов Матвей Сергеевич

группа P33111

Преподаватель:

Пашнин Александр Денисович

г. Санкт-Петербург, 2023

Оглавление

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.

Вариант:

Интерфейс передачи: **procfs**

Целевые структуры: **socket, memblock_type**

Ход работы

Программа на уровне ядра

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/fs.h>
#include <linux/memblock.h>
#include <linux/printk.h>
#include <net/net_namespace.h>
#include <net/sock.h>
#include <net/tcp.h>
#include <net/udp.h>
#include <net/udplite.h>
#include <net/raw.h>

#define BUFFER_SIZE 1024

static char proc_buffer[BUFFER_SIZE];
static struct proc_dir_entry *proc_entry;


static void format_bytes(unsigned long long bytes, char *buf, size_t buf_size) {
    const char *suffixes[] = {"B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB"};
    const int devider = 1024;
    int suffixIndex = 0;
    bytes = bytes * 10;

    while (bytes >= devider * 10 && suffixIndex <
sizeof(suffixes)/sizeof(suffixes[0])) {
        bytes = (bytes + devider / 2) / devider;
        suffixIndex++;
    }

    if (bytes % 10 == 0) {
        snprintf(buf, buf_size,"%llu%s", bytes / 10, suffixes[suffixIndex]);
    } else {
        snprintf(buf, buf_size, "%llu.%llu%s", bytes / 10, bytes % 10,
suffixes[suffixIndex]);
    }
```

```
}


static int write_memblok_type_info(void) {
    struct memblock_type mmb_type = memblock.memory;

    char readable_bytes[10];
    format_bytes(mmb_type.total_size, readable_bytes, 10);

    return snprintf(
        proc_buffer,
        BUFFER_SIZE,
        "Memblock Type:\n"
        "Symbolic name: %s\n"
        "Number of allocated regions: %lu\n"
        "Max number of regions: %lu\n"
        "Size of all regions: %s (%llu B)\n",
        mmb_type.name, mmb_type.cnt, mmb_type.max,
        readable_bytes, mmb_type.total_size
    );
}

static int write_socket_info(void) {
    struct net *net = get_net_ns_by_pid(1);

    return snprintf(
        proc_buffer,
        BUFFER_SIZE,
        "Socket:\n"
        "TCP: inuse: %d\n"
        "UDP: inuse: %d\n"
        "UDPLITE: inuse: %d\n"
        "RAW: inuse: %d\n",
        sock_prot_inuse_get(net, &tcp_prot),
        sock_prot_inuse_get(net, &udp_prot),
        sock_prot_inuse_get(net, &udplite_prot),
        sock_prot_inuse_get(net, &raw_prot)
    );
}
```

```
static ssize_t read_proc(struct file *filp, char *buffer, size_t length, loff_t *offset)
{
    int ret = 0;

    if (*offset > 0) {
        return 0;
    }

    if (strcmp(proc_buffer, "memblock_type") == 0) {
        write_memblok_type_info();
    } else if (strcmp(proc_buffer, "socket") == 0) {
        write_socket_info();
    } else {
        sprintf(proc_buffer, "Invalid argument\n");
    }

    ret = simple_read_from_buffer(buffer, length, offset, proc_buffer,
strlen(proc_buffer));

    return ret;
}

static ssize_t write_proc(struct file *filp, const char *buffer, size_t length, loff_t
*offset) {
    if (length > BUFFER_SIZE) {
        length = BUFFER_SIZE;
    }

    if (copy_from_user(proc_buffer, buffer, length)) {
        return -EFAULT;
    }

    proc_buffer[length] = '\0';

    return length;
}

static const struct proc_ops proc_fops = {
    .proc_read = read_proc,
    .proc_write = write_proc,
};
```

```
static int __init my_kernel_module_init(void) {
    proc_entry = proc_create("my_kernel_module", 0666, NULL, &proc_fops);
    if (proc_entry == NULL) {
        printk(KERN_ERR "Error creating /proc/my_kernel_module entry\n");
        return -ENOMEM;
    }

    printk(KERN_INFO "/proc/my_kernel_module created\n");

    return 0;
}

static void __exit my_kernel_module_exit(void) {
    proc_remove(proc_entry);
    printk(KERN_INFO "/proc/my_kernel_module removed\n");
}

module_init(my_kernel_module_init);
module_exit(my_kernel_module_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("joulin");
MODULE_DESCRIPTION("Kernel module for retrieving information about
memblock_type and socket structures");
```

Программа на пользовательском уровне

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFER_SIZE 1024

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Usage: %s [memblock_type/socket]\n", argv[0]);
        exit(1);
    }

    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);

    FILE *fp = fopen("/proc/my_kernel_module", "w");
    if (fp == NULL) {
        printf("Error opening /proc/my_kernel_module\n");
        exit(1);
    }

    fprintf(fp, "%s", argv[1]);
    fclose(fp);

    fp = fopen("/proc/my_kernel_module", "r");
    if (fp == NULL) {
        printf("Error opening /proc/my_kernel_module\n");
        exit(1);
    }

    while (fgets(buffer, BUFFER_SIZE, fp)) {
        printf("%s", buffer);
    }

    fclose(fp);

    return 0;
}
```

Makefile для сборки:

```
obj-m += kernel_module.o

reins:
	@echo "Reinstall module - START"
	make rm
	make build
	make ins
	@echo "Reinstall module - END"

build:  # Build kernel module
	@echo Build module in $(shell uname -r)
	make -C $(KERNEL_PATH)/lib/modules/$(shell uname -r)/build
M=$(PWD) modules

clean:
	@echo "Clean module files"
	make -C $(KERNEL_PATH)/lib/modules/$(shell uname -r)/build
M=$(PWD) clean

ins:   # Install kernel module
	@echo "Install module"
	sudo insmod kernel_module.ko

rm:
	@echo "Remove module"
	sudo rmmod kernel_module.ko

usr:   # Compile user program
	gcc user_program.c -o user_program
```

Результаты работы:

Переустановка модуля и компилирование пользовательской программы:

```
joulin@matthewserver:~/lab2$ make reins
Reinstall module - START
make rm
make[1]: Entering directory '/home/joulin/lab2'
Remove module
sudo rmmod kernel_module.ko
make[1]: Leaving directory '/home/joulin/lab2'
make build
make[1]: Entering directory '/home/joulin/lab2'
Build module in 6.7.1
make -C /lib/modules/6.7.1/build M=/home/joulin/lab2 modules
make[2]: Entering directory '/mnt/data/linux-6.7.1'
make[2]: Leaving directory '/mnt/data/linux-6.7.1'
make[1]: Leaving directory '/home/joulin/lab2'
make ins
make[1]: Entering directory '/home/joulin/lab2'
Install module
sudo insmod kernel_module.ko
make[1]: Leaving directory '/home/joulin/lab2'
Reinstall module - END
joulin@matthewserver:~/lab2$ make usr
gcc user_program.c -o user_program
joulin@matthewserver:~/lab2$ make clean
Clean module files
make -C /lib/modules/6.7.1/build M=/home/joulin/lab2 clean
make[1]: Entering directory '/mnt/data/linux-6.7.1'
  CLEAN   /home/joulin/lab2/Module.symvers
make[1]: Leaving directory '/mnt/data/linux-6.7.1'
```

Информация по socket

```
joulin@matthewserver:~/lab2$ ./user_program socket
Socket:
TCP: inuse: 3
UDP: inuse: 2
UDPLITE: inuse: 0
RAW: inuse: 0
```

Информация по memblock_type:

```
joulin@matthewserver:~/lab2$ ./user_program memblock_type
Memblock Type:
Symbolic name: memory
Number of allocated regions: 9
Max number of regions: 1024
Size of all regions: 4GB (4294967296 B)
```

Заключение

Лабораторная работа заставла действительно задуматься и разобравться в том, как работают и взаимодействуют между друг другом модули в Linux. В процессе выполнения для доступа к некоторым струкурам в ядре, которые не были экспортированы изначально, мне пришлось полностью пересобрать ядро с внесёнными мною измнениями. Это были незабываемые 6 часов компилирования ядра. Но по итогу, когда всё заработало я ощутил некоторое могущество и преисполнился. Было весело.

// TODO не забыть написать вывод