Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 4

Тема Виртуальная файловая система /proc

Студент Сушина А.Д.

Группа ИУ7-61б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва.
2020 г

Оглавление

1 Задание на лабораторную работу

Цель лабораторной работы: изучить работу файловой системы /proc.

Часть 1.

- В пользовательском режиме вывести на экран информацию об окружении процесса с комментариями;
- В пользовательском режиме вывести на экран информацию о состоянии процесса с комментариями;
- Вывести информацию из файла cmdline и директории fd.

Часть 2.

Написать загружаемый модуль ядра, создать файл в файловой системе proc, sysmlink, subdir. Используя соответствующие функции передать данные из пространства пользователя в пространство ядра (введенные данные вывести в файл ядра) и из пространства ядра в пространство пользователя. Продемонстрировать это

2 Листинг программ

На листингах 1-2 представлен код программ для 1 и второй части лабораторной работы.

Листинг 1. Часть 1. Вывод информации об окружении, состоянии процесса и информацию из cmdline и директории fd.

```
1. char *outputNames[] = {

2.     "pid",
3.     "filename",
4.     "state",
5.     "ppid",
6.     "gid",
7.     "session",
8.     "tty_nr",
9.     "tp_gid",
10.          "flags",
11.          "minflt",
12.          "cminflt",
13.          "majflt",
14.          "cmajflt",
15.          "utime",
16.          "stime",
17.          "cutime",
18.          "cstime",
19.          "priority",
20.          "nice",
21.          "num_threads",
22.          "itrealvalue",
23.          "start_tiime",
24.          "vsize",
25.          "rss",
26.          "rsslim",
```

```
        "startcode",
        "endcode",
        "startstack",
        "kstkesp",
        "kstkeip",
        "signal",
        "blocked",
        "sigignore",
        "sigcatch",
        "wchan",
        "nswap",
        "cnswap",
        "exit_signal",
        "processor",
        "rt_priority",
        "policy",
        "delayacct_blkio_tics",
        "quest_time",
        "cquest_time",
        "start_data",
        "end_data",
        "start_brk",
        "arg_start",
        "arg_end",
        "env_start",
        "env_end",
        "exit_code"

};


static int i = 0;

void statOutput(char *buf)
{
    int len = strlen(buf);
    int currentName = 0;
    char *pch = strtok(buf, " ");

    while (pch != NULL && i < 51)
    {
        printf("\n%15s:\t %s", outputNames[i], pch);
        pch = strtok(NULL, " ");
        i++;
    }
}

void simpleOutput(char *buf)
{
    printf("%s\n", buf);
}

void read_one_file(char* filename, void (*print_func)(char*))
{
    char buf[BUF_SIZE];
    int i, len;
    FILE *f = fopen(filename, "r");
    while ((len = fread(buf, 1, BUF_SIZE, f)) > 0)
    {
        for (i = 0; i < len; i++)
```

```
                if( buf[i] == 0)
                    buf[i] = 10;
            buf[len - 1] = 0;
            print_func(buf);
        }
        fclose(f);
    }

    int main(int argc, char *argv[])
    {
        printf("\n_____________________________\n");
        printf("STAT \n\n");
        read_one_file("/proc/self/stat", statOutput);
        printf("\n_____________________________\n");
        printf("ENVIRON\n\n");
        read_one_file("/proc/self/environ",  simpleOutput);
        printf("\n_____________________________\n");
        printf("CMDLINE\n\n");
        read_one_file("/proc/self/cmdline",  simpleOutput);
        printf("\n_____________________________\n");
        printf("FD\n\n");
        execl("/bin/ls", "ls", "/proc/self/fd", NULL);
        return 0;
    }
```

Листинг 2. Часть 2. Модуль ядра fortune

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/vmalloc.h>
#include <linux/uaccess.h>
#include <linux/sched.h>
#include <linux/init_task.h>

    MODULE_LICENSE("GPL");
    MODULE_DESCRIPTION("Fortune Cookie Kernel Module");

    #define COOKIE_BUF_SIZE PAGE_SIZE
    #define TEMP_BUF_SIZE 256

    ssize_t fortune_read(struct file *file, char *buf, size_t count,
loff_t *f_pos);
    ssize_t fortune_write(struct file *file, const char *buf, size_t
count, loff_t *f_pos);

    int fortune_init(void);
    void fortune_exit(void);

    struct file_operations fops = {
        .owner = THIS_MODULE,
        .read = fortune_read,
        .write = fortune_write,
    };

    static char *cookie_buf;
```

```
24.     static struct proc_dir_entry *proc_entry;
25.     static unsigned read_index;
26.     static unsigned write_index;

27.     char temp[TEMP_BUF_SIZE];

28.     struct task_struct *task = &init_task;

29.     int len;

30.     ssize_t fortune_read(struct file *file, char *buf, size_t count,
    loff_t *f_pos)
31.     {
32.         if (*f_pos > 0)
33.             return 0;

34.         if (read_index >= write_index)
35.             read_index = 0;

36.         len = 0;
37.         if (write_index > 0)
38.         {
39.             len = sprintf(temp, "%s\n", &cookie_buf[read_index]);

40.             copy_to_user(buf, temp, len);
41.             buf += len;
42.             read_index += len;
43.         }

44.         *f_pos += len;

45.         return len;
46.     }

47.     ssize_t fortune_write(struct file *file, const char *buf, size_t
    count, loff_t *f_pos)
48.     {
49.         int space_available = (COOKIE_BUF_SIZE - write_index) + 1;

50.         if (count > space_available)
51.         {
52.             printk(KERN_INFO "+_+ cookie pot is full\n");
53.             return -ENOSPC;
54.         }

55.         if (copy_from_user(&cookie_buf[write_index], buf, count))
56.             return -EFAULT;

57.         write_index += count;
58.         cookie_buf[write_index - 1] = 0;

59.         return count;
60.     }

61.     int fortune_init(void)
62.     {
```

```
63.          cookie_buf = (char *) vmalloc(COOKIE_BUF_SIZE);

64.          if (!cookie_buf)
65.          {
66.              printk(KERN_INFO "+_+ not enough memory for the cookie pot\
   n");
67.              return -ENOMEM;
68.          }

69.          memset(cookie_buf, 0, COOKIE_BUF_SIZE);
70.          proc_entry = proc_create("fortune", 0666, NULL, &fops);

71.          if (!proc_entry)
72.          {
73.              vfree(cookie_buf);
74.              printk(KERN_INFO "+_+ Couldn't create proc entry\n");
75.              return -ENOMEM;
76.          }

77.          read_index = 0;
78.          write_index = 0;

79.          proc_mkdir("my_dir_in_proc", NULL);
80.          proc_symlink("my_symbolic_in_proc", NULL, "/proc/fortune");

81.          printk(KERN_INFO "+_+ fortune module loaded.\n");
82.          return 0;
83.      }

84.      void fortune_exit(void)
85.      {
87.          if (cookie_buf)
88.              vfree(cookie_buf);

90.      }

91.      module_init(fortune_init);
92.      module_exit(fortune_exit);
86.          remove_proc_entry("fortune", NULL);

89.          printk(KERN_INFO "+_+ fortune module unloaded.\n");
```

3 Результаты работы программы

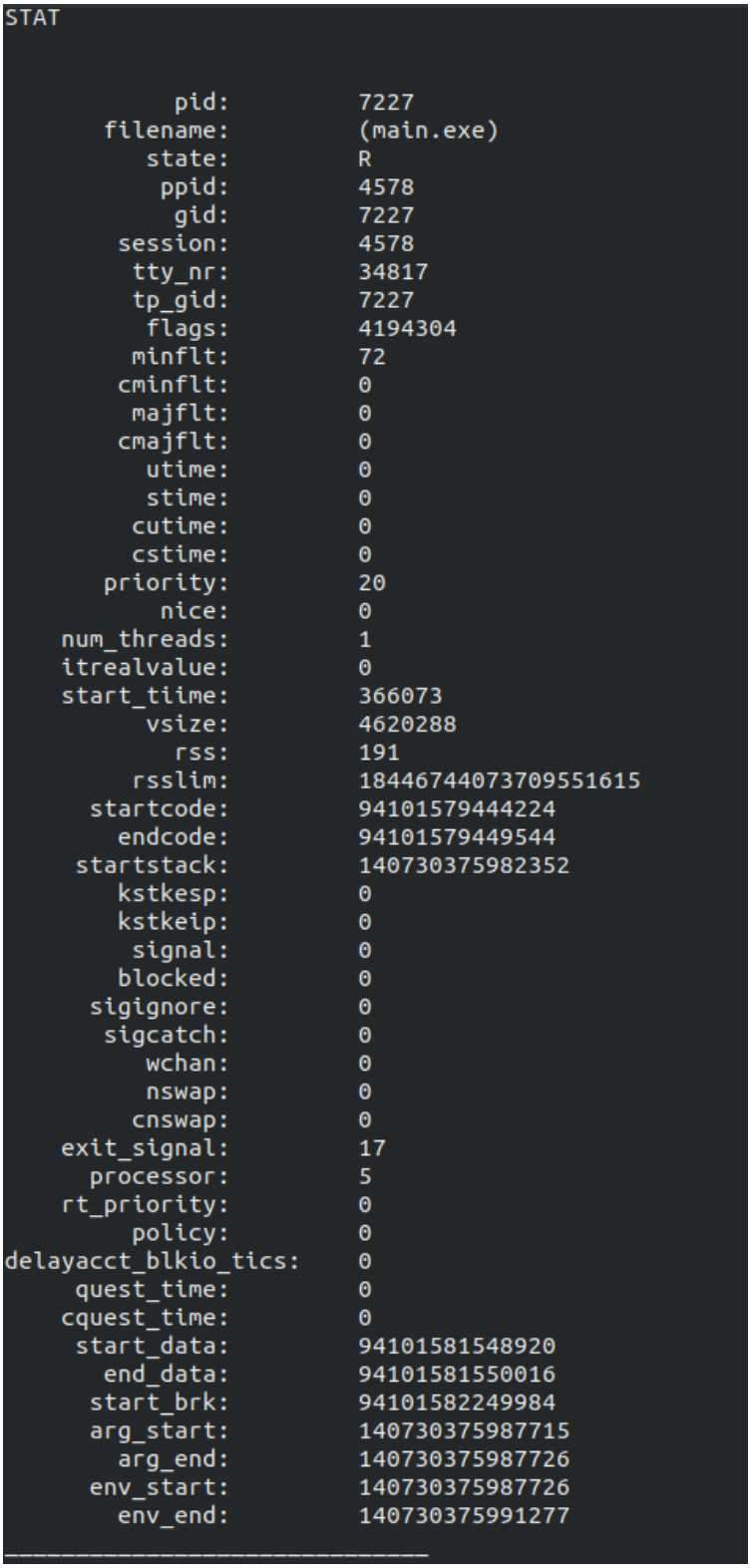

Рис 1. Вывод программы из части 1. Содержание файла STAT.

Рис 2. Вывод программы из части 1. Содержимое файла ENVIRON

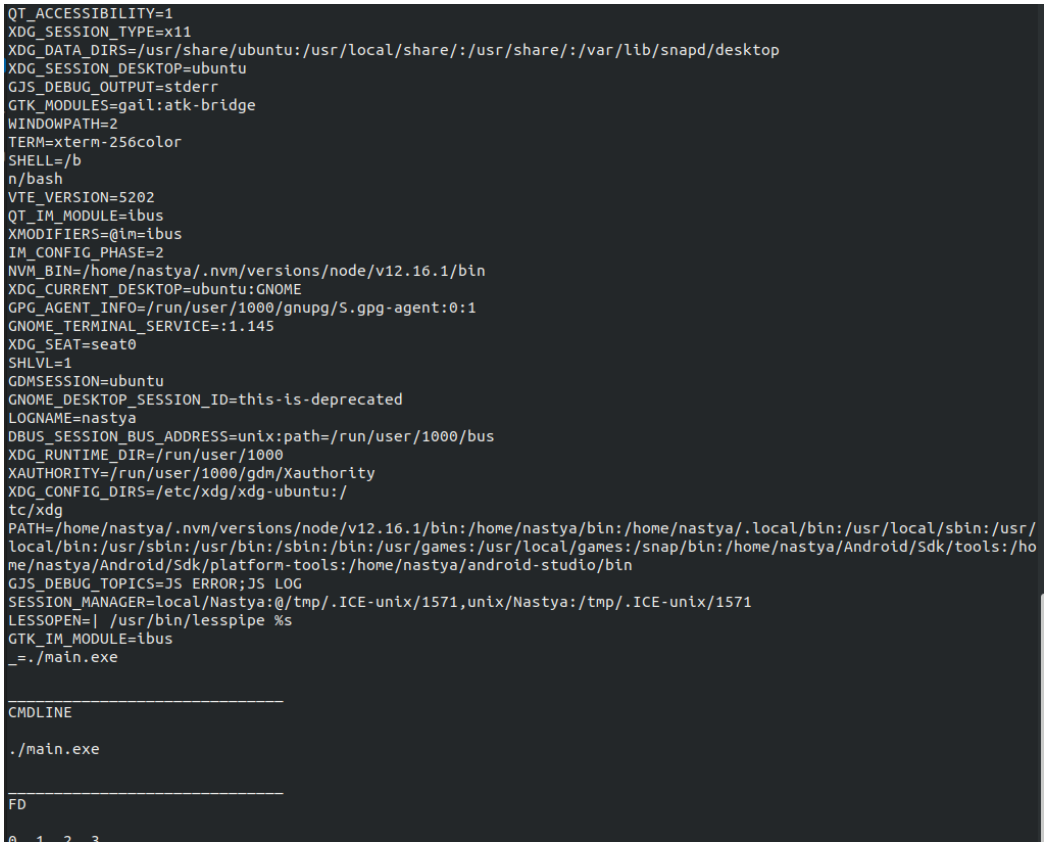

Рис 3. Вывод программы из части 1. Содержимое файла Environ, cmdline и директории fd.

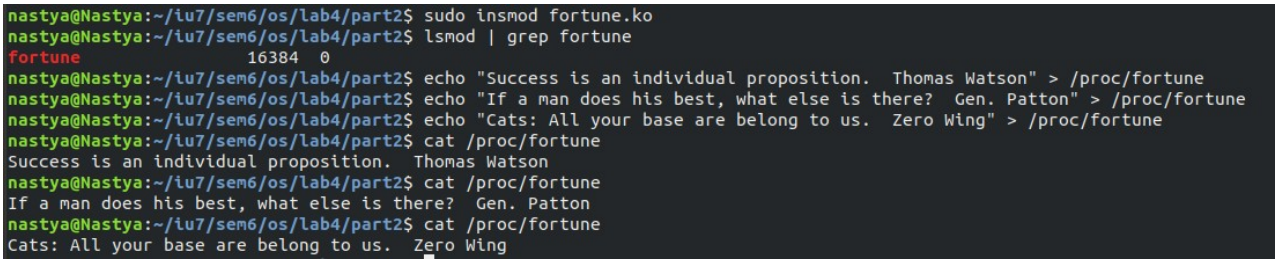

Рис 4. Демонстрация работы загружаемого модуля ядра Fortune