**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
## По курсу «Операционные системы»

**Тема** Виртуальная файловая система /proc

**Студент** Неклепаева А. Н.

**Группа** ИУ7-63б

**Преподаватель** Рязанова Н. Ю.

Москва.
2020 г.

**Задание № 1:**

1. В пользовательском режиме вывести на экран информацию об окружении процесса

с комментариями;

2. В пользовательском режиме вывести на экран информацию о состоянии процесса с

комментариями;

3. Вывести информацию из файла cmdline и директории fd.

Листинг программы к заданию 1

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
#include <unistd.h>

#define BUF_SIZE 0x100

int read_env()
{
    char buf[BUF_SIZE];
    int len, i;
    FILE *f = fopen("/proc/self/environ", "r");

    if (!f)
    {
        printf("env: fopen err\n");
        return -1;
    }

    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;
        printf("%s", buf);
    }

    if (fclose(f))
    {
        printf("env: fclose err\n");
        return -1;
    }

    return 0;
}

int read_cmdline()
{
    char buf[BUF_SIZE];
    int len, i;
    FILE *f = fopen("/proc/self/cmdline", "r");

    if (!f)
```

```
    {
        printf("cmdline: fopen err\n");
        return -1;
    }

    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;
        printf("%s", buf);
    }

    if (fclose(f))
    {
        printf("cmdline: fclose err\n");
        return -1;
    }

    return 0;
}

int read_stat()
{
    char *records[] = {"pid", "comm", "state", "ppid", "pgrp", "session",
"tty_nr", "tpgid", "flags", "minflt",
    "cminflt", "majflt", "cmajflt", "utime", "stime", "cutime", "cstime",
"priority", "nice", "num_threads",
    "itrealvalue", "starttime", "vsize", "rss", "rsslim", "startcode",
"endcode", "startstack", "kstkesp",
    "kstkeip", "signal", "blocked", "sigignore", "sigcatch", "wchan", "nswap",
"cnswap", "exit_signal",
    "processor", "rt_priority", "policy", "delayacct_blkio_ticks",
"guest_time", "cguest_time", "start_data",
    "end_data", "start_brk", "arg_start", "arg_end", "env_start", "env_end",
"exit_code"};

    int i;
    char buf[BUF_SIZE];
    FILE *f = fopen("/proc/self/stat", "r");

    if (!f)
    {
        printf("stat: fopen err\n");
        return -1;
    }

    fread(buf, 1, BUF_SIZE, f);
    char *pch = strtok(buf, " ");

    i = 0;
    while (pch != NULL)
    {
        printf("%s = %s\n", records[i], pch);
        pch = strtok(NULL, " ");
        i++;
    }

    if (fclose(f))
    {
        printf("stat: fclose err\n");
        return -1;
```

```
    }

    return 0;
}

int read_fd()
{
    struct dirent *dirp;
    DIR *dp;
    char str[BUF_SIZE];
    char path[BUF_SIZE];
    dp = opendir("/proc/self/fd");//открыть директорию

    if (!dp)
    {
        printf("fd: opendir err\n");
        return -1;
    }

    while ((dirp = readdir(dp)) != NULL)//читаем директорию
    {
        if ((strcmp(dirp->d_name, ".") != 0) &&
        (strcmp(dirp->d_name, "..") != 0))
        {
            sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
            readlink(path, str, BUF_SIZE);
            printf("%s -> %s\n", dirp->d_name, str);
        }
    }

    if (closedir(dp))
    {
        printf("fd: closedir err\n");
        return -1;
    }

    return 0;
}

int main(int argc, char *argv[])
{
    printf("ENVIRON\n");
    if (read_env())
        return -1;

    printf("\n\nCMDLINE\n");
    if (read_cmdline())
        return -1;

    printf("\n\nSTAT\n");
    if (read_stat())
        return -1;

    printf("\n\nFD\n");
    if (read_fd())
        return -1;

    return 0;
}
```

Рис 1: вывод на экран информации об окружении процесса

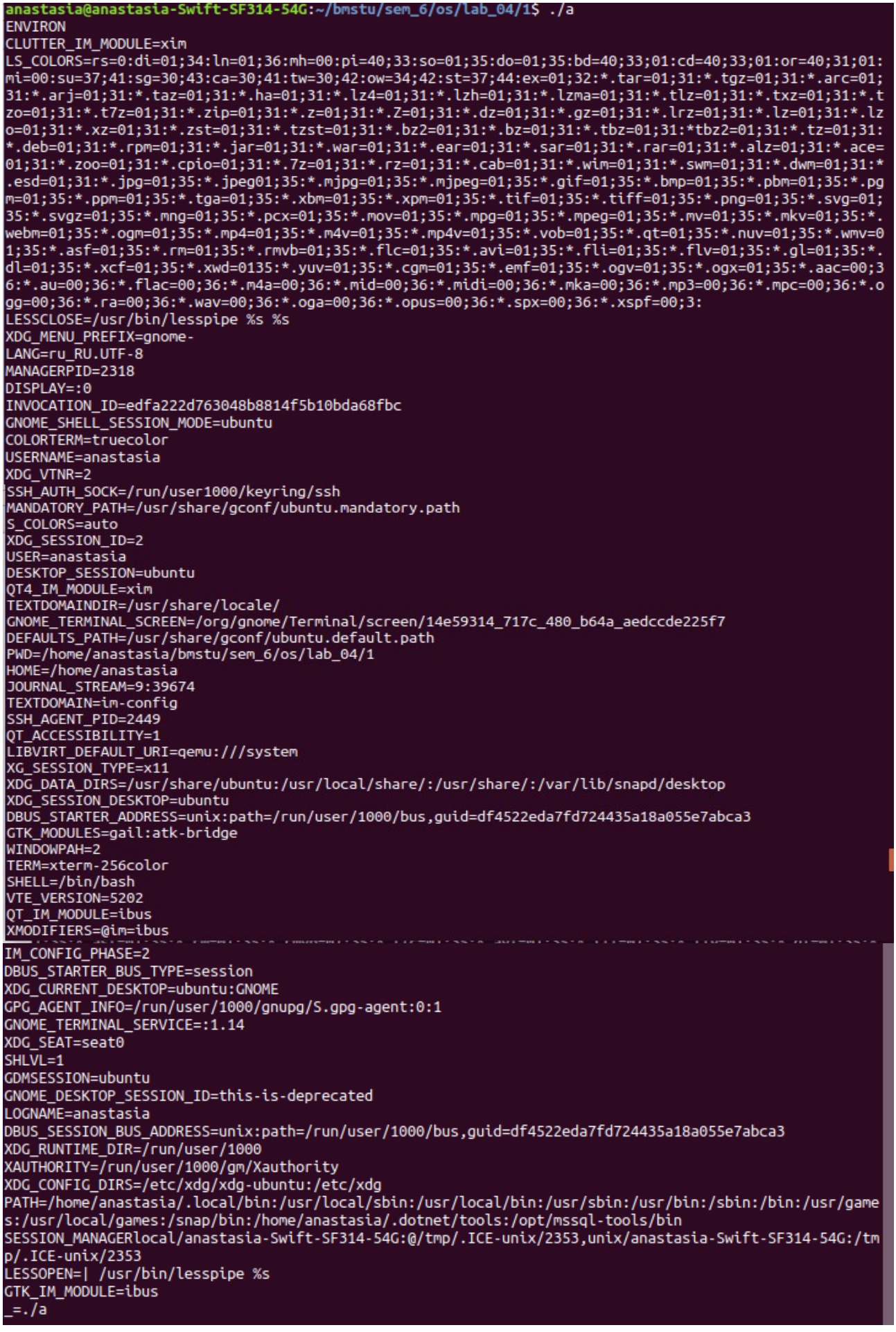

Рис 2: вывод на экран содержания файла cmdline

```
CMDLINE
./a
```

Рис 3: вывод на экран информации о состоянии процесса

```
STAT
pid = 4432
comm = (a)
state = R
ppid = 4420
pgrp = 4432
session = 4420
tty_nr = 34817
tpgid = 4432
flags = 4194304
minflt = 74
cminflt = 0
majflt = 0
cmajflt = 0
utime = 0
stime = 0
cutime = 0
cstime = 0
priority = 20
nice = 0
num_threads = 1
itrealvalue = 0
starttime = 155622
vsize = 4620288
rss = 187
rsslim = 18446744073709551615
startcode = 94829593735168
endcode = 94829593741768
startstack = 140724610809936
kstkesp = 0
kstkeip = 0
signal = 0
blocked = 0
sigignore = 0
sigcatch = 0
wchan = 0
nswap = 0
cnswap = 0
exit_signal = 17
processor = 1
rt_priority = 0
policy = 0
delayacct_blkio_ticks = 0
guest_time = 0
cguest_time = 0
start_data = 94829595839832
end_data = 94829595840960
start_brk = 94829624168448
arg_start = 140724610814543
arg_end = 14072p?p
```

Рис 4: вывод на экран содержания директории fd

```
FD
0 -> /dev/pts/1?<?V
1 -> /dev/pts/1?<?V
2 -> /dev/pts/1?<?V
3 -> /proc/4432/fdV
```

**Задание № 2**

Написать программу – загружаемый модуль ядра (LKM) – которая поддерживает чтение из пространства пользователя и запись в пространство пользователя. После загрузки модуля пользователь может загружать в него строки с помощью команды echo, а затем автоматически считывать их с помощью команды cat. В программе необходимо создать поддиректорию и символическую ссылку.

Листинг к заданию 2

```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/vmalloc.h>
#include <linux/uaccess.h>

#define BUFSIZE 100
#define MAX_COOKIE_LENGTH PAGE_SIZE

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR( "Neklepaeva A. N.");

static struct proc_dir_entry *proc_entry;
static struct proc_dir_entry *proc_link;
static struct proc_dir_entry *proc_dir;
static char *cookie_pot = NULL;
int cookie_index;
int next_fortune;

static ssize_t fortune_write(struct file *file, const char __user *ubuf, size_t
count, loff_t *ppos)
{
    int free_space = (MAX_COOKIE_LENGTH - next_fortune) + 1;

    if (count > free_space)
    {
        printk(KERN_INFO "Cookie pot full.\n");
        return -ENOSPC;
    }

    if (copy_from_user(&cookie_pot[cookie_index], ubuf, count))
    {
        return -EFAULT;
    }

    cookie_index += count;
    cookie_pot[cookie_index - 1] = 0;

    return count;
}

static ssize_t fortune_read(struct file *file, char __user *ubuf, size_t count,
loff_t *ppos)
{
    int len;
    char buf[256];

    if (cookie_index == 0 || *ppos > 0)
```

```
    {
        return 0;
    }

    if (next_fortune >= cookie_index)
    {
        next_fortune = 0;
    }

    len = sprintf(buf, "%s\n", &cookie_pot[next_fortune]);
    copy_to_user(ubuf, buf, len);
    next_fortune += len;
    *ppos += len;

    return len;
}

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

static int __init fortune_init(void)
{
    cookie_pot = (char *)vmalloc(MAX_COOKIE_LENGTH);
    if (!cookie_pot)
    {
        printk(KERN_INFO "Not enough memory for the cookie pot.\n");
        return -ENOMEM;
    }

    memset( cookie_pot, 0, MAX_COOKIE_LENGTH );

    proc_entry = proc_create("fortune", 0666, NULL, &myops);
    proc_dir = proc_mkdir("mydir", NULL);

    if (proc_entry == NULL)
    {
        vfree(cookie_pot);
        printk(KERN_INFO "fortune: Couldn't create proc entry\n");
        return -ENOMEM;
    }

    proc_link = proc_symlink("link", NULL, "fortune");

    cookie_index = 0;
    next_fortune = 0;

    printk(KERN_INFO "fortune: Module inited.\n");

    return 0;
}

static void __exit fortune_exit(void)
{
    remove_proc_entry("fortune", NULL);
    remove_proc_entry("mydir", NULL);
    remove_proc_entry("link", NULL);

    if (cookie_pot)
        vfree(cookie_pot);
```

```
    printk(KERN_INFO "fortune: Module exited.\n");
}

module_init(fortune_init);
module_exit(fortune_exit);
```

Рис 5: демонстрация работы «fortune cookie» LKM, демонстрация передачи данных из пространства пользователя и из пространства ядра в пространство пользователя

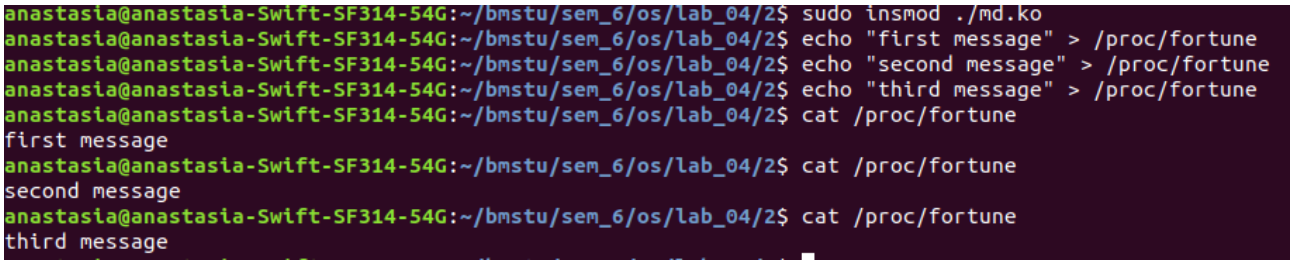


Рис 6: вывод ls -al /proc , демонстрация создания
файла fortune, символьной ссылки link и директории mydir в файловой системе proc