|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

|  |  |
| --- | --- |
| **Дисциплина** Операционные системы  **Тема** \_Виртуальная файловая система /proc\_  **Студент** \_Ильясов И. М.\_  **Группа** \_ИУ7-63Б\_  **Оценка (баллы)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Преподаватель** \_Рязанова Н. Ю.\_ |  |

Москва, 2020 г.

**Задание 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.

Ниже на рисунке 1 приведен результат вывода на экран информации об окружении процесса (/proc/self/environ).

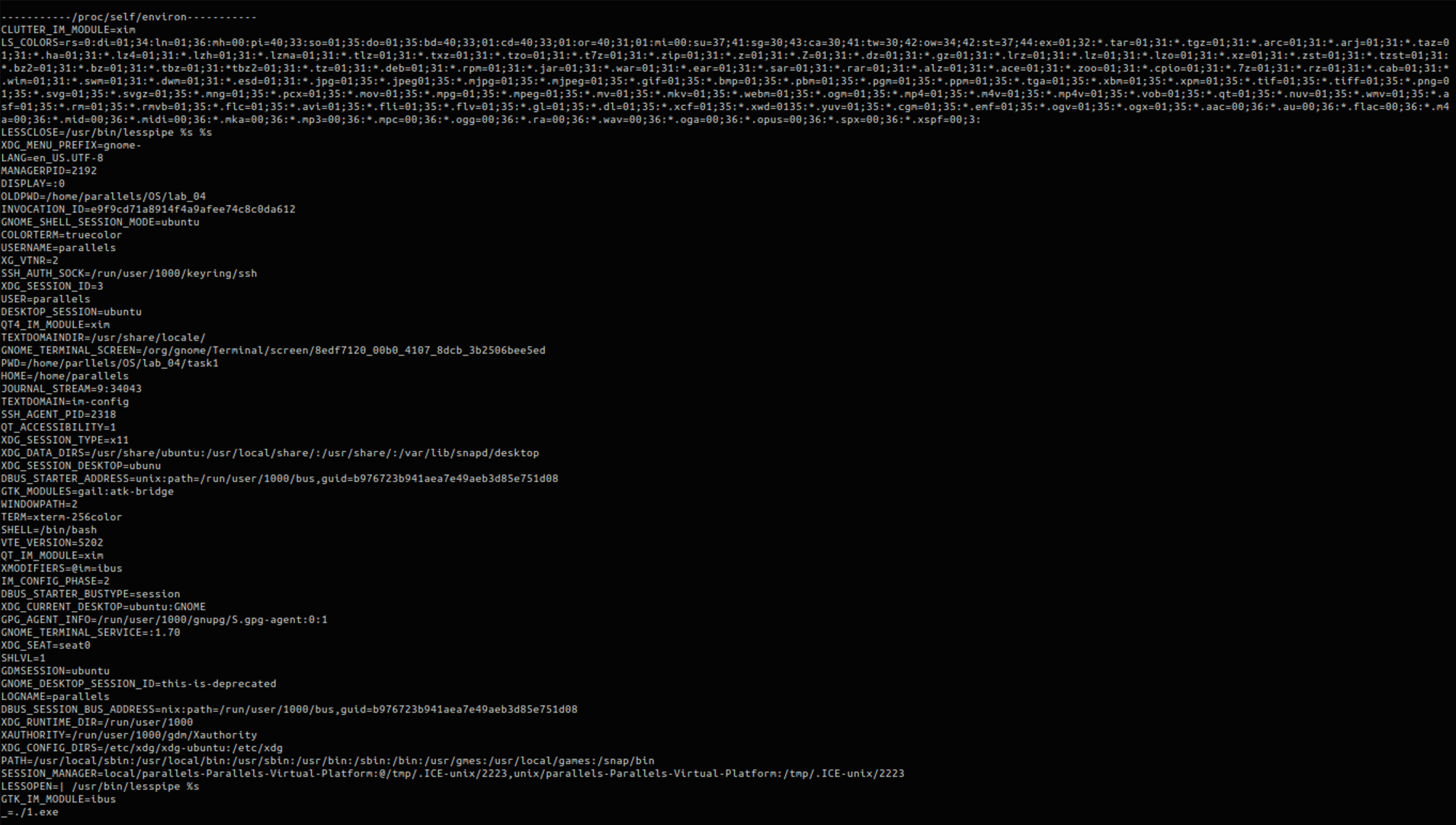


Рисунок 1. Информация об окружении процесса (/proc/self/environ).

На рисунке 2 приведен результат вывода на экран информации об состоянии процесса (/proc/self/stat).

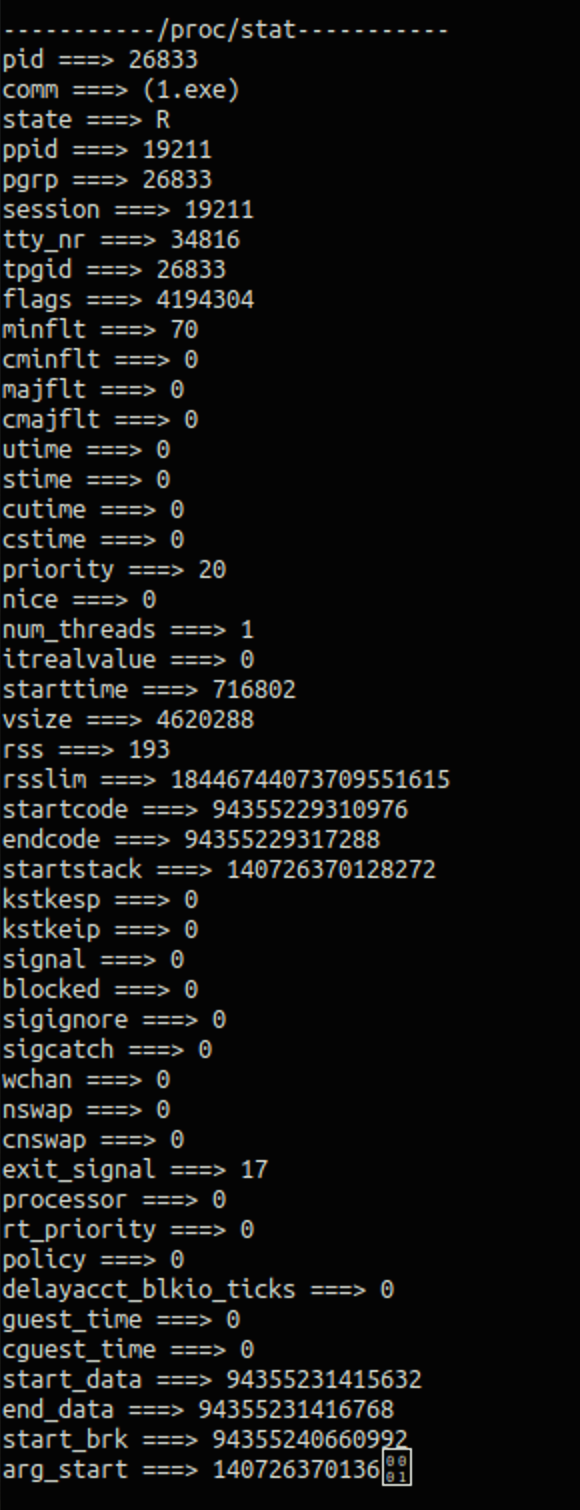


Рисунок 2. Информация об состоянии процесса (/proc/self/stat).

На рисунке 3 приведен результат вывода на экран содержания директории fd (/proc/self/fd).

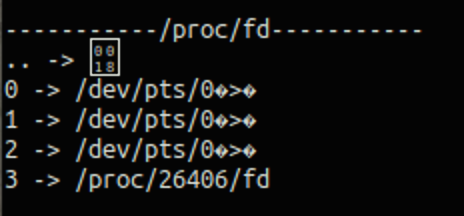


Рисунок 3. Содержание директории fd (/proc/self/fd).

На рисунке 4 приведен результат вывода на экран содержания файла cmdline (/proc/self/cmdline).

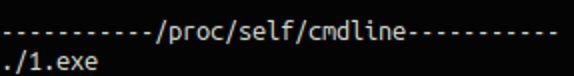


Рисунок 4. Содержание файла cmdline (/proc/self/cmdline).

Далее приведен листинг программы из задания №1:

#include <stdio.h>

#include <string.h>

#include <dirent.h>

#include <unistd.h>

#define BUF\_SIZE 0x100

#define SUCCESS 0

#define FILE\_ERROR -1

#define FREAD\_ERROR -2

#define SPRINTF\_ERROR -3

#define READLINK\_ERROR -4

#define OPEN\_DIR\_ERROR -5

#define CLOSE\_DIR\_ERROR -6

char \*attr[] = {"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 print\_file(char \*name)

{

char buf[BUF\_SIZE];

int len = 0;

FILE \*f = NULL;

f = fopen(name, "r");

if (!f)

return FILE\_ERROR;

printf("\n-----------%s-----------\n", name);

while ((len = fread(buf, 1, BUF\_SIZE, f)) > 0) { for (int i = 0; i < len; i++)

{

if (buf[i] == 0)

{

buf[i] = 10;

}

}

buf[len - 1] = 0;

printf("%s", buf); }

if (fclose(f) != 0)

return FILE\_ERROR;

printf("\n");

return SUCCESS;

}

int print\_stat()

{

char buf[BUF\_SIZE];

FILE \*f = NULL;

char \*pch = NULL;

int i = 0;

f = fopen("/proc/self/stat", "r");

if (!f)

return FILE\_ERROR;

if (fread(buf, 1, BUF\_SIZE, f) <= 0)

return FREAD\_ERROR;

printf("\n-----------stat-----------\n");

pch = strtok(buf, " ");

while (pch != NULL)

{

printf("%s ===> %s\n", attr[i], pch);

pch = strtok(NULL, " ");

i++;

}

if (fclose(f) != 0)

return FILE\_ERROR;

return SUCCESS;

}

int print\_fd()

{

struct dirent \*dirp = NULL;

DIR \*dp = NULL;

char str[BUF\_SIZE];

char path[BUF\_SIZE];

dp = opendir("/proc/self/fd");

if (!dp)

return OPEN\_DIR\_ERROR;

printf("\n-----------fd-----------\n");

while ((dirp = readdir(dp)) != NULL)

{

if ((strcmp(dirp->d\_name, ".") != 0) && (strcmp(dirp->d\_name, ".") != 0))

{

if (sprintf(path, "%s%s", "/proc/self/fd/", dirp->d\_name) < 0)

return SPRINTF\_ERROR;

readlink(path, str, BUF\_SIZE);

printf("%s -> %s\n", dirp->d\_name, str);

}

}

if (closedir(dp) < 0)

return CLOSE\_DIR\_ERROR;

return SUCCESS;

}

int main(int argc, char \*\*argv)

{

int err = 0;

if ((err = print\_file("/proc/self/environ")))

return err;

if ((err = print\_stat()))

return err;

if ((err = print\_fd()))

return err;

if ((err = print\_file("/proc/self/cmdline")))

return err;

return err;

}

**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.

На рисунке 5 представлен частичный вывод команды ls для демонстрации создания файла (mydev), символьной ссылки (link) и директории (mydir) в файловой системе proc.

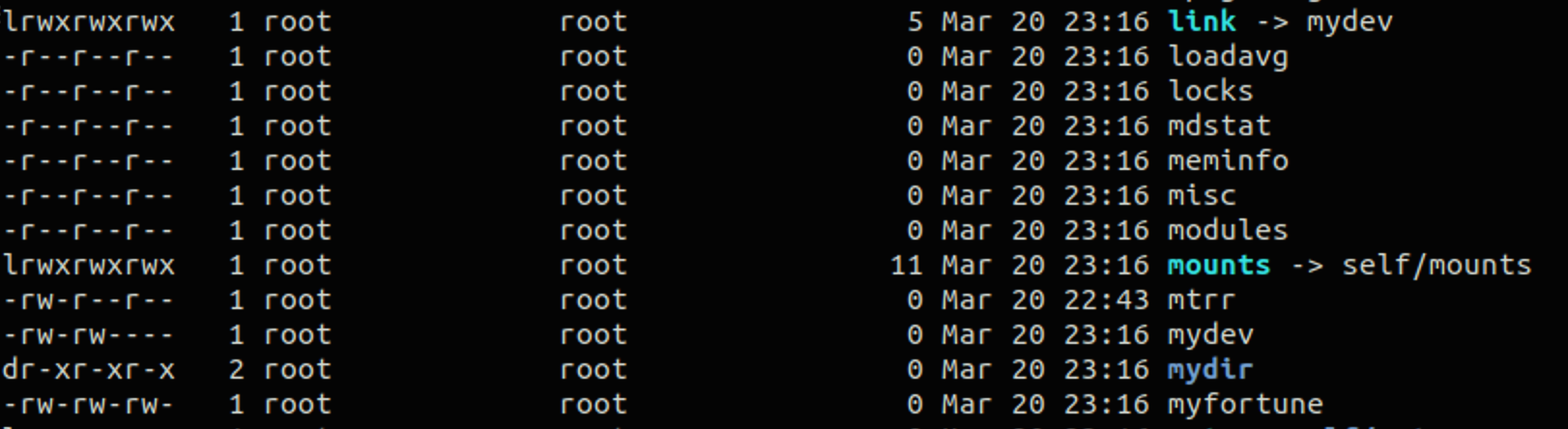


Рисунок 5. Частичный вывод ls.

Далее приведен листинг модуля, где создается файл, символьная ссылка и директория в файловой системе proc.

#include <linux/module.h>

#include <linux/moduleparam.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/proc\_fs.h>

#define BUFSIZE 100

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_AUTHOR("Ilyasov Idris");

static struct proc\_dir\_entry \* ent;

static ssize\_t mywrite(struct file \*file, const char \_\_user \*ubuf,size\_t count, loff\_t \*ppos)

{

printk(KERN\_DEBUG "write handler\n");

return -1;

}

static ssize\_t myread(struct file \*file, char \_\_user \*ubuf,size\_t count, loff\_t \*ppos)

{

printk(KERN\_DEBUG "read handler\n");

return 0;

}

static struct file\_operations myops =

{

.owner = THIS\_MODULE,

.read = myread,

.write = mywrite

};

static int simple\_init(void)

{

ent = proc\_create("mydev", 0660, NULL, &myops);

ent = proc\_symlink("link", NULL, "mydev");

ent = proc\_mkdir("mydir", NULL);

return 0;

}

static void simple\_cleanup(void)

{

proc\_remove(ent);

}

module\_init(simple\_init);

module\_exit(simple\_cleanup);

Ниже представлен результат работы программы, в которой данные передаются из пространства пользователя и из пространства ядра в пространство пользователя.

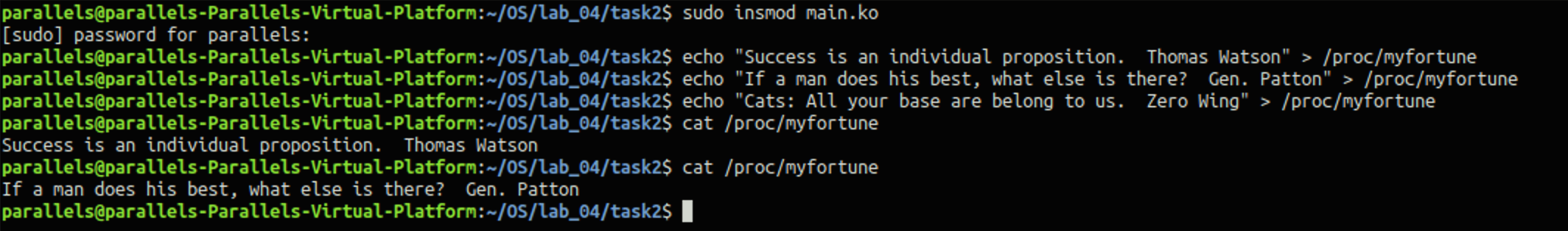


Рисунок 6. Пример результата работы программы, в которой данные передаются из пространства пользователя и из пространства ядра в пространство пользователя.

Далее приведен листинг программы, в которой данные передаются из пространства пользователя и из пространства ядра в пространство пользователя.

#include <linux/module.h>

#include <linux/moduleparam.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>

#define MAX\_COOKIE\_LENGTH PAGE\_SIZE

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_AUTHOR("Ilyasov Idris");

static struct proc\_dir\_entry \*proc\_entry;

char \*cookie\_pot;

int cookie\_index;

int next\_fortune;

char buf[256];

static ssize\_t fortune\_write(struct file \*file, const char \_\_user \*ubuf,size\_t count, loff\_t \*ppos)

{

if (count > MAX\_COOKIE\_LENGTH - cookie\_index + 1)

{

printk(KERN\_DEBUG "Big count\n");

return -ENOSPC;

}

if (copy\_from\_user(cookie\_pot + cookie\_index, ubuf, count))

{

printk(KERN\_DEBUG "Copy\_from\_user error\n");

return -EFAULT;

}

cookie\_index += count;

cookie\_pot[cookie\_index - 1] = 0;

printk(KERN\_DEBUG "Good write\n");

return count;

}

static ssize\_t fortune\_read(struct file \*file, char \_\_user \*ubuf,size\_t count, loff\_t \*ppos)

{

int len = 0;

if (\*ppos > 0)

return 0;

if (next\_fortune >= cookie\_index)

{

next\_fortune = 0;

}

if (cookie\_index > 0)

{

len = sprintf(buf, "%s\n", cookie\_pot + next\_fortune);

copy\_to\_user(ubuf, buf, len);

next\_fortune += len;

ubuf += len;

printk(KERN\_DEBUG "Len: %d\n", len);

\*ppos += len;

}

return len;

}

static struct file\_operations myops =

{

.owner = THIS\_MODULE,

.read = fortune\_read,

.write = fortune\_write

};

static int simple\_init(void)

{

cookie\_pot = (char \*)vmalloc(MAX\_COOKIE\_LENGTH);

if (!cookie\_pot)

return -ENOMEM;

memset(cookie\_pot, 0, MAX\_COOKIE\_LENGTH);

proc\_entry = proc\_create("myfortune", 0666, NULL, &myops);

if (proc\_entry == NULL)

{

vfree(cookie\_pot);

printk(KERN\_DEBUG "fortune: Couldn't create proc entry\n");

return -ENOMEM;

}

cookie\_index = 0;

next\_fortune = 0;

printk(KERN\_DEBUG "fortune: Init\n");

proc\_symlink("link", NULL, "mydev");

proc\_mkdir("mydir", NULL);

return 0;

}

static void simple\_cleanup(void)

{

proc\_remove(proc\_entry);

vfree(cookie\_pot);

printk(KERN\_DEBUG "fortune: Clean\n");

}

module\_init(simple\_init);

module\_exit(simple\_cleanup);