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»

ОТЧЁТ

По лабораторной работе №4

По курсу: «Операционные системы»

Тема: «Виртуальная файловая система /proc»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Преподаватель: Рязанова Н. Ю.

Москва

2020

Часть 1

Задание. Используя виртуальную файловую систему `proc`, вывести информацию об окружении процесса, о состоянии процесса, информацию из файла `cmdline` и директории `fd`.

В листинге 1 представлена программа `proc.c`, выводящая информацию о процессе. На рисунках 1–3 продемонстрирована работа программы.

Листинг 1: task1/proc.c

```c
#include <errno.h>
#include <dirent.h>
#include <linux/limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUF_SIZE 4096

FILE *open_proc_file(int pid, const char *filename);
void close_proc_file(FILE *file);
void print_proc_file(FILE *file, char delim);
void print_proc_environ(FILE *file, int pid);
void print_proc_cmdline(FILE *file, int pid);
void print_proc_stat(FILE *file);
void print_proc_fd(int pid);

int main(int argc, char *argv[])
{
        if (argc > 2) {
                fprintf(stderr, "Usage: %s [pid=self]\n", argv[0]);
                return EXIT_FAILURE;
        }

        const int pid = argc == 1 ? getpid() : atoi(argv[1]);

        FILE *file = NULL;

        file = open_proc_file(pid, "environ");
        print_proc_environ(file, pid);
        close_proc_file(file);

        file = open_proc_file(pid, "stat");
        print_proc_stat(file);
        close_proc_file(file);

        file = open_proc_file(pid, "cmdline");
        print_proc_cmdline(file, pid);
        close_proc_file(file);

        print_proc_fd(pid);
}

/*
 * Открытие файла /proc/<pid>/<filename>
 */
FILE *open_proc_file(int pid, const char *filename)
{
        char pathname[PATH_MAX];
        snprintf(pathname, sizeof pathname, "/proc/%d/%s", pid, filename);

        FILE *file = fopen(pathname, "r");
        if (file == NULL) {
                fprintf(stderr, "fopen(\"%s\", \"r\"): %s\n", pathname, strerror(errno));
                exit(EXIT_FAILURE);
        }

        return file;
}

/*
 * Закрытие файла
 */
```

```c
void close_proc_file(FILE *file)
{
        fclose(file);
}

/*
 * Вывод на экран файла file с заменой символа '\0' на символ delim
 */
void print_proc_file(FILE *file, char delim)
{
        char buf[BUF_SIZE];
        size_t len;

        while ((len = fread(buf, 1, BUF_SIZE, file)) > 0) {
                for (char *c = buf; c < buf + len; ++c) {
                        if (*c == '\0') {
                                *c = delim;
                        }
                }
                buf[len - 1] = '\0';
                printf("%s", buf);
        }
}

/*
 * Вывод информации об окружении процесса
 */
void print_proc_environ(FILE *file, int pid)
{
        printf("=== Environment list of the process with id %d:\n", pid);
        print_proc_file(file, '\n');
}

/*
 * Вывод информации из файла cmdline
 */
void print_proc_cmdline(FILE *file, int pid)
{
        printf("\n=== Command line for the process with id %d:\n", pid);
        print_proc_file(file, ' ');
}

/*
 * Вывод информации о состоянии процесса
 */
void print_proc_stat(FILE *file)
{
        static char *fields[] = {
                "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", NULL
        };

        char buf[BUF_SIZE];
        const size_t len = fread(buf, 1, BUF_SIZE, file);
        buf[len - 1] = '\0';

        printf("\n\n=== State of the process:\n");
        for (
                char *pch = strtok(buf, " "), **pfield = fields;
                pch && *pfield;
                pch = strtok(NULL, " "), ++pfield
        ) {
                printf("%-21s %s\n", *pfield, pch);
        }
}

/*
 * Вывод информации из директории fd
 */
void print_proc_fd(int pid)
{
        char dirname[PATH_MAX];
```

```c
        snprintf(dirname, sizeof dirname, "/proc/%d/fd", pid);

        DIR *dir = opendir(dirname);
        if (dir == NULL) {
                fprintf(stderr, "opendir(\"%s\"): %s\n", dirname, strerror(errno));
                exit(EXIT_FAILURE);
        }

        printf("\n\n=== List of open files for the process with id %d:\n", pid);
        struct dirent *entry = NULL;
        while ((entry = readdir(dir)) != NULL) {
                if (strcmp(entry->d_name, ".") == 0
                 || strcmp(entry->d_name, "..") == 0) {
                        continue;
                }

                char path[PATH_MAX], str[PATH_MAX];
                snprintf(path, sizeof path, "%s/%s", dirname, entry->d_name);

                const int n = readlink(path, str, sizeof str);
                str[n] = '\0';

                printf("\t%-17s -> %s\n", path, str);
        }

        closedir(dir);
}
```

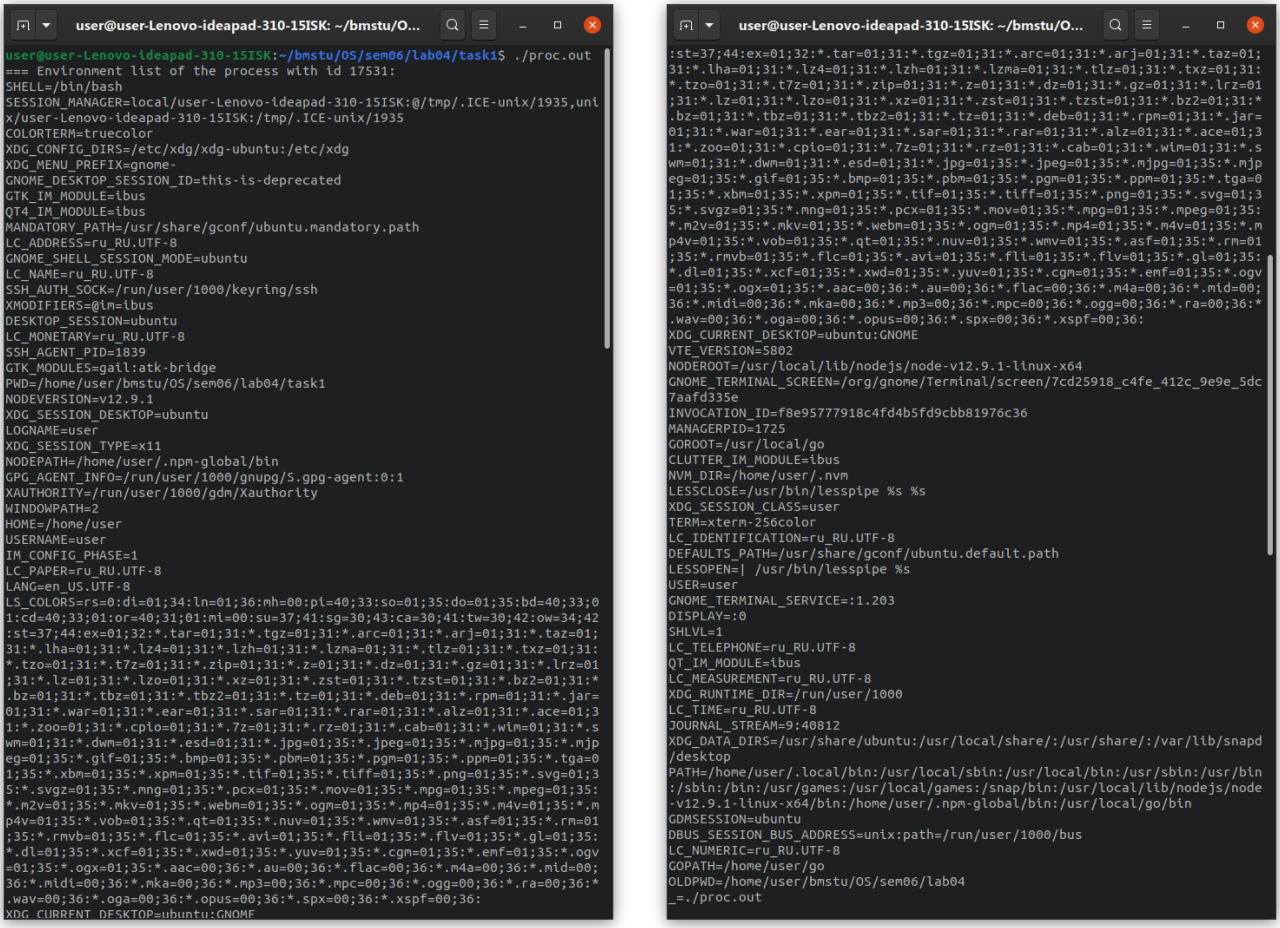

Рис. 1: Информация об окружении процесса

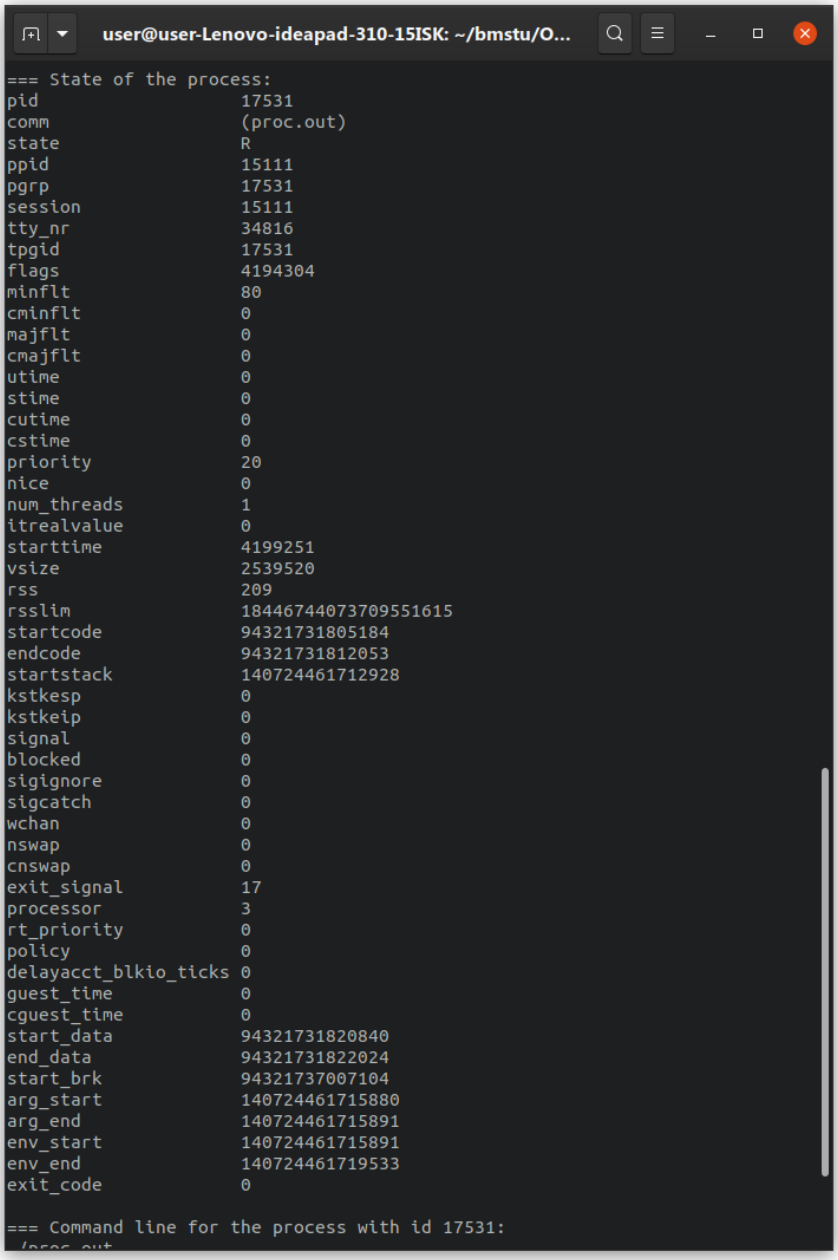

Рис. 2: Информация о состоянии процесса

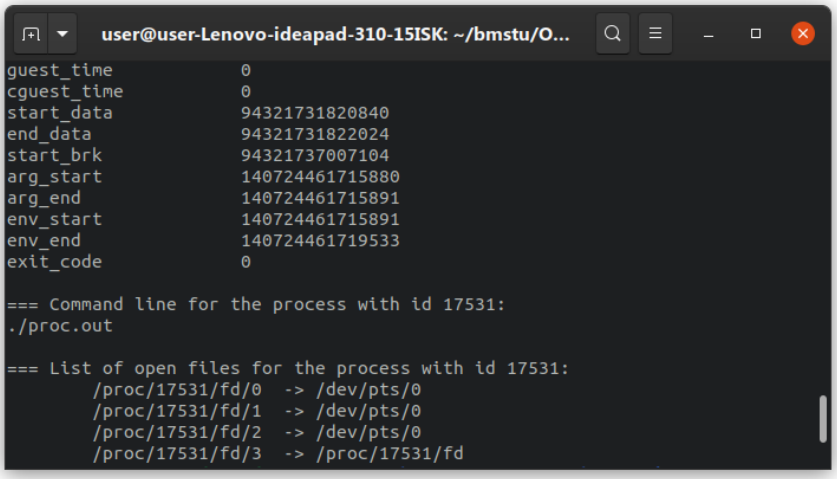

Рис. 3: Информация из файла `cmdline` и директории `fd`

Часть 2

Задание. Написать загружаемый модуль ядра, создать файл в файловой системе `proc`, поддиректорию и символическую ссылку. Используя соответствующие функции, передать данные из пространства пользователя в пространство ядра (введённые данные вывести в файл ядра) и из пространства ядра в пространство пользователя. Продемонстрировать это.

В листинге 2 содержится код программы загружаемого модуля. На рисунке 4 продемонстрирована работа программы. На рисунке 5 показаны созданные в `/proc` файлы после загрузки модуля.

Листинг 2: task2/fortune.c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kerimov A. IU7-64b");
MODULE_DESCRIPTION("LKM fortune");

#define MAX_COOKIE_LENGTH PAGE_SIZE

#define FORTUNE_DIRNAME  "fortune_dir"
#define FORTUNE_FILENAME "fortune_file"
#define FORTUNE_SYMLINK  "fortune"
#define FORTUNE_FILEPATH FORTUNE_DIRNAME "/" FORTUNE_FILENAME

static struct proc_dir_entry *fortune_dir = NULL;
static struct proc_dir_entry *fortune_file = NULL;
static struct proc_dir_entry *fortune_symlink = NULL;

static char *cookie_pot;  // Хранилище наших фортунок
static int cookie_index;  // Индекс для добавления печеньки
static int next_fortune;  // Индекс для чтения печеньки

static char tmp[MAX_COOKIE_LENGTH];

ssize_t fortune_read(struct file *filep, char __user *buf, size_t count, loff_t *offp)
{
        int len;

        if (*offp > 0 || cookie_index == 0)
                return 0;

        if (next_fortune >= cookie_index)
                next_fortune = 0;

        len = snprintf(tmp, MAX_COOKIE_LENGTH, "%s\n", &cookie_pot[next_fortune]);
        if (copy_to_user(buf, tmp, len)) {
                printk(KERN_ERR "=== fortune: copy_to_user error\n");
                return -EFAULT;
        }
        next_fortune += len;
        *offp += len;

        return len;
}

ssize_t fortune_write(struct file *file, const char __user *buf, size_t len, loff_t *offp)
{
        if (len > MAX_COOKIE_LENGTH - cookie_index + 1) {
                printk(KERN_ERR "=== fortune: cookie_pot overflow error\n");
                return -ENOSPC;
        }

        if (copy_from_user(&cookie_pot[cookie_index], buf, len)) {
                printk(KERN_ERR "=== fortune: copy_to_user error\n");
                return -EFAULT;
        }
```

```
62
63          cookie_index += len;
64          cookie_pot[cookie_index - 1] = '\0';
65
66          return len;
67  }
68
69  static struct file_operations fops = {
70          owner: THIS_MODULE,
71          read:  fortune_read,
72          write: fortune_write
73  };
74
75  static void cleanup(void)
76  {
77          if (fortune_symlink) remove_proc_entry(FORTUNE_SYMLINK, NULL);
78          if (fortune_file)    remove_proc_entry(FORTUNE_FILENAME, fortune_dir);
79          if (fortune_dir)     remove_proc_entry(FORTUNE_DIRNAME, NULL);
80          if (cookie_pot)      vfree(cookie_pot);
81  }
82
83  static int shutdown_enomem(const char *s)
84  {
85          cleanup();
86          printk(KERN_ERR "=== fortune: %s error\n", s);
87          return -ENOMEM;
88  }
89
90  static int __init fortune_init(void)
91  {
92          if (!(cookie_pot = vmalloc(MAX_COOKIE_LENGTH)))
93                  return shutdown_enomem("vmalloc");
94          memset(cookie_pot, 0, MAX_COOKIE_LENGTH);
95
96          if (!(fortune_dir = proc_mkdir(FORTUNE_DIRNAME, NULL)))
97                  return shutdown_enomem("proc_create");
98          if (!(fortune_file = proc_create(FORTUNE_FILENAME, 0666, fortune_dir, &fops)))
99                  return shutdown_enomem("proc_create");
100         if (!(fortune_symlink = proc_symlink(FORTUNE_SYMLINK, NULL, FORTUNE_FILEPATH)))
101                 return shutdown_enomem("proc_symlink");
102
103         cookie_index = 0;
104         next_fortune = 0;
105
106         printk(KERN_INFO "=== fortune: module loaded\n");
107         return 0;
108 }
109
110 static void __exit fortune_exit(void)
111 {
112         cleanup();
113         printk(KERN_INFO "=== fortune: module unloaded\n");
114 }
115
116 module_init(fortune_init)
117 module_exit(fortune_exit)
```

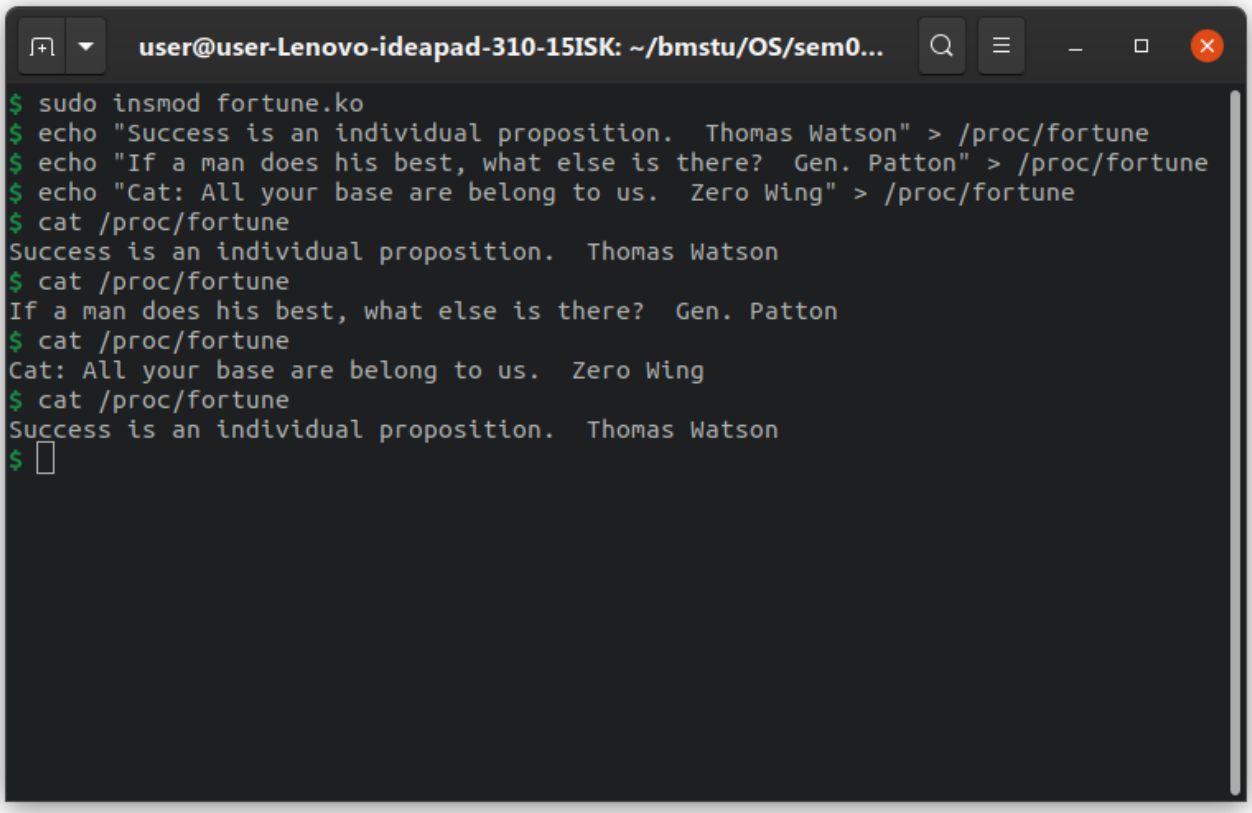

Рис. 4: Демонстрация работы модуля `fortune`

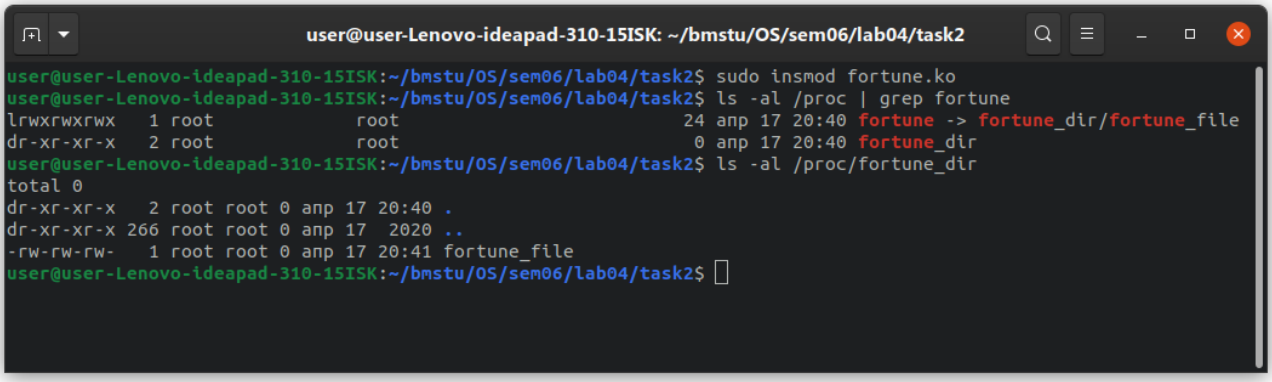

Рис. 5: Созданные файлы в `/proc` после загрузки модуля