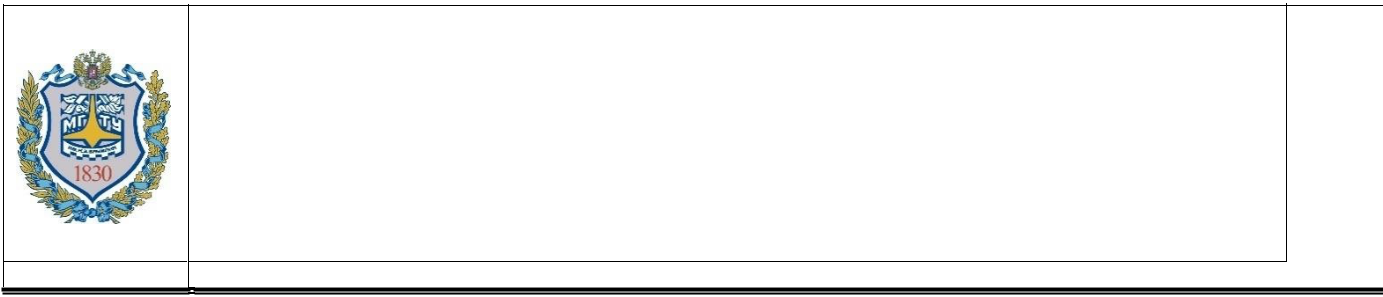
**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«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 №4.2**

**Группа:** ИУ7-62Б

**Студент:** Блохин Д.М.

**Оценка (баллы):**

**Преподаватель:** Рязанова Н.Ю.

*Москва, 2021*

**Задание**

Написать программу – загружаемый модуль ядра (LKM) – которая поддерживает чтение из пространства пользователя и запись в пространство пользователя из пространства ядра.

После загрузки модуля пользователь должен иметь возможность загружать в него строки с помощью команды echo, а затем считывать их с помощью команды cat.

**Содержимое файла md.c:**

#include<linux/module.h>

#include<linux/proc\_fs.h>

#include<linux/slab.h>

#include<linux/vmalloc.h>

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Blokhin Dmitry");

#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\_release(struct inode \*node, struct file \*file);

int fortune\_open(struct inode \*node, struct file \*file);

int fortune\_init(void);

void

fortune\_exit(void);

static const struct proc\_ops pops = {

.proc\_read = fortune\_read,

.proc\_write = fortune\_write,

.proc\_release = fortune\_release,

.proc\_open = fortune\_open,

};

static char \*cookie\_buf;

static struct proc\_dir\_entry

\*proc\_entry; static unsigned

read\_index;

static unsigned write\_index;

char temp[TEMP\_BUF\_SIZE]; struct

task\_struct \*task = &init\_task; int

len, t\_len;

int fortune\_init(void)

{

cookie\_buf = (char \*) vmalloc(COOKIE\_BUF\_SIZE); if (!cookie\_buf)

{

printk(KERN\_INFO "not enough memory\n");

return -ENOMEM;

}

memset(cookie\_buf, 0, COOKIE\_BUF\_SIZE);

proc\_entry = proc\_create("fortune", 0666, NULL, &pops); if (!proc\_entry)

{

vfree(cookie\_buf);

printk(KERN\_INFO "Couldn't create proc entry\n"); return -ENOMEM;

}

read\_index = 0;

write\_index = 0;

proc\_mkdir("my\_dir\_fortune", NULL);

proc\_symlink("my\_symbolic\_fortune", NULL, "/proc/fortune"); printk(KERN\_INFO "fortune module loaded.\n");

return 0;

}

ssize\_t fortune\_read(struct file \*file, char \*buf, size\_t count, loff\_t \*f\_pos) {

if (\*f\_pos > 0 || write\_index ==

0) return 0;

if (read\_index >=

write\_index) read\_index = 0;

len = copy\_to\_user(buf, "%s\n", &cookie\_buf[read\_index]); read\_index += len;

\*f\_pos += len;

printk(KERN\_INFO "proc called read \n"); return len;

}

int fortune\_release(struct inode \*node, struct file \*file)

{

printk(KERN\_INFO "proc called release \n");

return 0;

}

int fortune\_open(struct inode \*node, struct file \*file)

{

printk(KERN\_INFO "proc called open \n"); return 0;

}

ssize\_t fortune\_write(struct file \*file, const char \*buf, size\_t count, loff\_t \*f\_pos) {

int space\_available = (COOKIE\_BUF\_SIZE - write\_index) + 1;

if (count > space\_available)

{

printk(KERN\_INFO "+\_+ buf is full\n");

return -ENOSPC;

}

if (copy\_from\_user(&cookie\_buf[write\_index], buf, count))

return -EFAULT;

write\_index += count;

cookie\_buf[write\_index - 1] = 0;

printk(KERN\_INFO "proc called write \n");

return count;

}

void fortune\_exit(void)

{

remove\_proc\_entry("fortune", NULL);

if (cookie\_buf)

vfree(cookie\_buf);

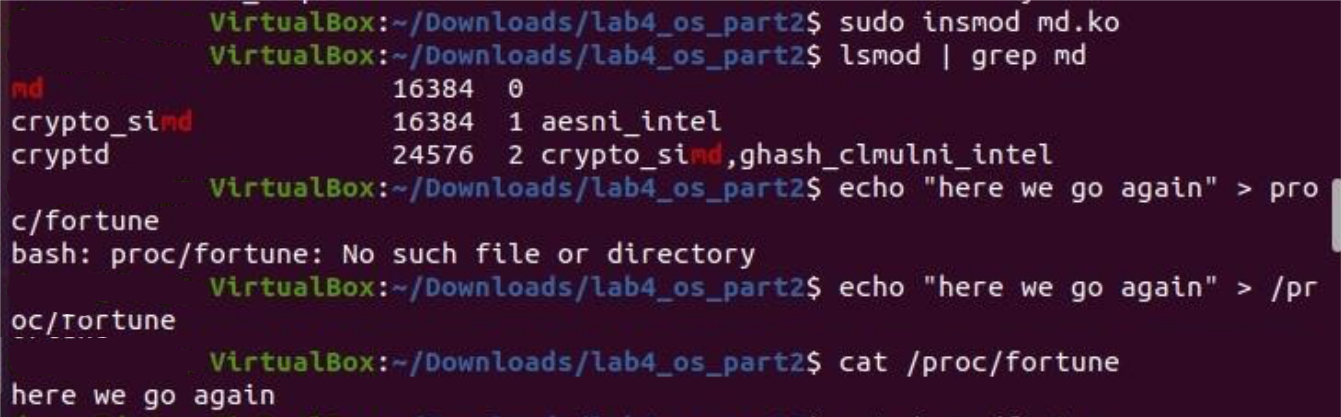
printk(KERN\_INFO "fortune module unloaded.\n");

}

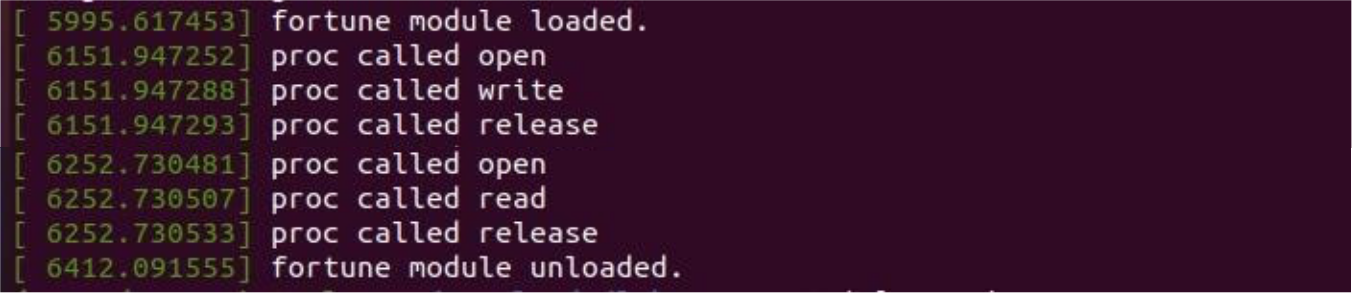
module\_init(fortune\_init);

module\_exit(fortune\_exit);

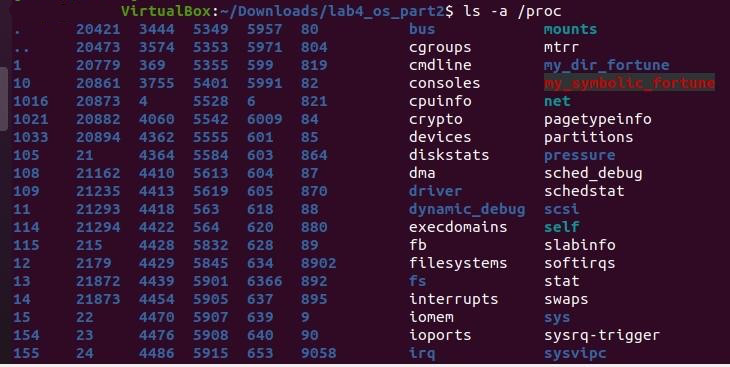
**Загрузка модуля – проверка загрузки – демонстрация работы загружаемого модуля – выгрузка модуля:**



**Содержимое файла /var/log/syslog:**



**Вывод содержимого /proc (с созданной директорией -- my\_dir\_fortune и символической ссылкой -- my\_symbolic\_fortune):**



**РЕАЛИЗАЦИЯ ПРИ ПОМОЩИ SEQUENCE**

**Содержимое файла fortune\_seq.c:**

#include<linux/module.h>

#include<linux/proc\_fs.h>

#include<linux/slab.h>

#include<linux/vmalloc.h>

#include<linux/seq\_file.h>

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Blokhin Dmitry");

static char \*str = NULL;

unsigned int

write\_index; unsigned int

read\_index;

#define COOKIE\_POT\_SIZE PAGE\_SIZE

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

ssize\_t my\_write(struct file\* file, const char user \*buffer, size\_t count, loff\_t

\*f\_pos); int my\_release(struct inode \*node, struct file \*file); int my\_open(struct inode \*inode, struct file \*file);

static const struct proc\_ops pops = {

.proc\_read = seq\_read,

.proc\_write = my\_write,

.proc\_release = my\_release,

.proc\_open = my\_open,

};

static int init fortune\_init(void)

{

printk(KERN\_INFO "fortune init\n");

write\_index = 0;

read\_index = 0;

str = vmalloc(COOKIE\_POT\_SIZE); if

(!str)

{

printk(KERN\_INFO "Error: can't malloc cookie buffer\n"); return -ENOMEM;

}

memset(str, 0, COOKIE\_POT\_SIZE);

proc\_entry = proc\_create("fortune", S\_IRUGO | S\_IWUGO, NULL, &pops); if(!proc\_entry)

{

vfree(str);

printk(KERN\_INFO "Error: can't create fortune file\n"); return -ENOMEM;

}

proc\_dir = proc\_mkdir("fortune\_dir", NULL);

if(!proc\_dir)

{

vfree(str);

printk(KERN\_INFO "Error: can't create fortune

file\n"); return -ENOMEM;

}

proc\_sym = proc\_symlink("fortune\_symlink", NULL, "/proc/fortune"); if(!proc\_sym)

{

vfree(str);

printk(KERN\_INFO "Error: can't create fortune

file\n"); return -ENOMEM;

}

printk(KERN\_INFO "fortune: Fortune module loaded successfully\n"); return 0;

}

static int my\_show(struct seq\_file \*m, void \*v)

{

printk(KERN\_INFO "fortune called my\_show\n");

seq\_printf(m, "%s", str + read\_index);

int len = strlen(str + read\_index);

if (len)

read\_index += len + 1;

return 0;

}

|  |  |  |
| --- | --- | --- |
| ssize\_t my\_write(struct file\* file, const char | | user \*buffer, size\_t count, loff\_t \*f\_pos) |
| { |  |  |

printk(KERN\_INFO "fortune called my\_write\n");

if (copy\_from\_user(&str[write\_index], buffer, count))

return -EFAULT;

write\_index += count;

str[write\_index-1] =

0; return count;

}

int my\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "fortune called my\_open\n");

return single\_open(file, my\_show, NULL);

}

int my\_release(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "fortune called my\_release\n"); return single\_release(inode, file);

}

static void exit fortune\_exit(void)

{

printk(KERN\_INFO "fortune exit\n");

if (proc\_entry)

remove\_proc\_entry("fortune", NULL); if

(proc\_dir)

remove\_proc\_entry("fortune\_dir", NULL); if

(proc\_sym)

remove\_proc\_entry("fortune\_symlink", NULL); if (str)

vfree(str);

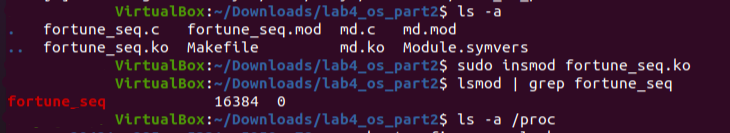
printk(KERN\_INFO "fortune: Fortune module unloaded\n");

}

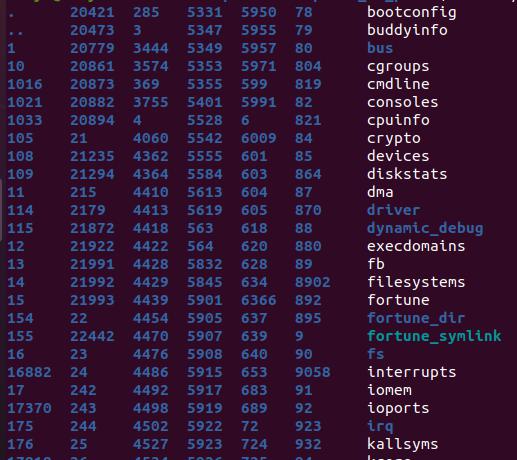
module\_init(fortune\_init);

module\_exit(fortune\_exit);

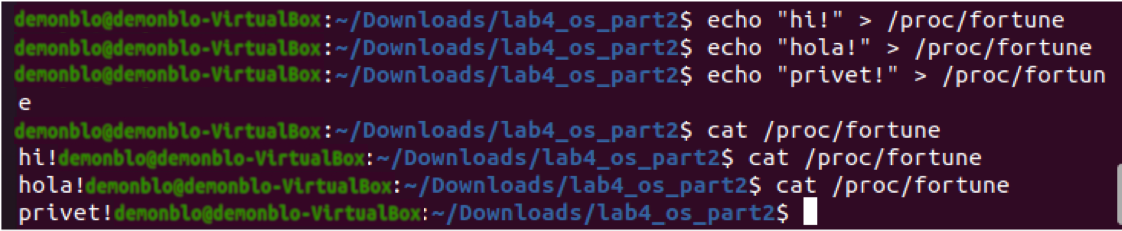
**Загрузка модуля**



**Вывод содержимого /proc, в том числе созданной директории и символической ссылки (fortune\_dir и fortune\_symlink):**



**Демонстрация работы загруженного модуля:**



**Содержимое файла /var/log/syslog:**

