|  |  |
| --- | --- |
|  | **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»

**ОТЧЕТ**

*к лабораторной работе №4*

*По курсу: «Операционные системы»*

*На тему: «Файловая система /proc»*

Студент ИУ7-66Б

Дедич Б.Б

Преподаватель

Рязанова Н.Ю.

*Москва, 2021 г.*

**Часть 1.**

# Листинг

|  |
| --- |
|  |

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <dirent.h>

#include <string.h>

#include "daemonize.h"

#define LOG\_FILE "/root/iu7-66B/Lab4/part1/log.txt"

#define BUF\_SIZE 0x100

struct print\_info

**{**

char name**[**NAME\_MAX**];**

int ignore\_newline**;**

void **(\***print\_file**)(**FILE **\*);**

void **(\***print\_name**)(**char **\*);**

**};**

void **\***get\_proc\_filename**(**char **\***name**,** int pid**,** const char **\***shortname**);**

void print\_proc\_file**(**int pid**,** struct print\_info **\***info**);**

void print\_proc\_stat**(**FILE **\*);**

void print\_proc\_statm**(**FILE **\*);**

void print\_proc\_mem**(**FILE **\*);**

void print\_proc\_fd**(**char **\*);**

void print\_symlink**(**char **\*);**

void print\_fields**(**FILE **\*,** char **\*\***fields**);**

FILE **\***f**;**

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

**{**

daemonize**(**"DAEMON"**);**

f **=** fopen**(**LOG\_FILE**,** "w"**);**

char buf**[**BUF\_SIZE**];**

int pid**;**

struct print\_info files**[][**8**]** **=** **{**

**{**"cmdline"**},**

**{**"cwd"**,** 0**,** **NULL,** print\_symlink**},**

**{**"environ"**},**

// {"exe", 1},

**{**"exe"**,** 0**,** **NULL,** print\_symlink**},**

**{**"fd"**,** 0**,** **NULL,** print\_proc\_fd**},**

**{**"maps"**},**

**{**"mem"**},**

**{**"root"**,** 0**,** **NULL,** print\_symlink**},**

**{**"stat"**,** 0**,** print\_proc\_stat**},**

**{**"statm"**,** 0**,** print\_proc\_statm**},**

**};**

int n\_files **=** **sizeof(**files**)** **/** **sizeof(**files**[**0**]);**

**if** **(**argc **!=** 2 **||** **(**pid **=** atoi**(**argv**[**1**]))** **<=** 0**)**

pid **=** getpid**();**

fprintf**(**f**,** "PID = %d\n"**,** pid**);**

**for** **(**int i **=** 0**;** i **<** n\_files**;** i**++)**

**{**

print\_proc\_file**(**pid**,** files**[**i**]);**

**}**

fclose**(**f**);**

**return** 0**;**

**}**

void **\***get\_proc\_filename**(**char **\***name**,** int pid**,** const char **\***shortname**)**

**{**

sprintf**(**name**,** "/proc/%d/%s"**,** pid**,** shortname**);**

**}**

void print\_proc\_file**(**int pid**,** struct print\_info **\***info**)**

**{**

fprintf**(**f**,** "\n===== %s =====\n"**,** info**->**name**);**

char name**[**PATH\_MAX**];**

get\_proc\_filename**(**name**,** pid**,** info**->**name**);**

**if** **(**info**->**print\_name**)**

**return** info**->**print\_name**(**name**);**

FILE **\***file **=** fopen**(**name**,** "r"**);**

**if** **(!**file**)**

**return;**

**if** **(**info**->**print\_file**)**

**{**

info**->**print\_file**(**file**);**

fclose**(**file**);**

**return;**

**}**

char buf**[**BUF\_SIZE**];**

int len**,** i**;**

**while** **((**len **=** fread**(**buf**,** 1**,** BUF\_SIZE**,** file**))** **>** 0**)**

**{**

**if** **(!**info**->**ignore\_newline**)**

**for** **(**i **=** 0**;** i **<** len**;** i**++)**

**if** **(**buf**[**i**]** **==** 0**)**

buf**[**i**]** **=** 10**;** // '\n'

buf**[**len**]** **=** 0**;**

fprintf**(**f**,** "%s"**,** buf**);**

**}**

fclose**(**file**);**

**}**

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};**

print\_fields**(**file**,** fields**);**

**}**

void print\_proc\_statm**(**FILE **\***file**)**

**{**

static char **\***fields**[]** **=** **{**

"size"**,** "resident"**,** "shared"**,** "text"**,** "lib"**,** "data"**,** "dt"**,** **NULL};**

print\_fields**(**file**,** fields**);**

**}**

void print\_proc\_fd**(**char **\***name**)**

**{**

DIR **\***dir **=** opendir**(**name**);**

**if** **(!**dir**)**

**{**

fprintf**(**stderr**,** "opendir('%s'): %s\n"**,** name**,** strerror**(**errno**));**

exit**(**EXIT\_FAILURE**);**

**}**

int flag **=** 1**;**

char path**[**PATH\_MAX**];**

struct dirent **\***dirp **=** **NULL;**

**while** **(**flag **&&** **(**dirp **=** readdir**(**dir**)))**

**{**

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

**{**

flag **=** 1**;**

**}**

**else** **if** **(**strcmp**(**dirp**->**d\_name**,** ".."**)** **==** 0**)**

**{**

flag **=** 1**;**

**}**

**else**

**{**

snprintf**(**path**,** **sizeof(**path**),** "%s/%s"**,** name**,** dirp**->**d\_name**);**

print\_symlink**(**path**);**

**}**

**}**

closedir**(**dir**);**

**}**

void print\_symlink**(**char **\***path**)**

**{**

char str**[**PATH\_MAX**];**

const int n **=** readlink**(**path**,** str**,** **sizeof(**str**));**

str**[**n**]** **=** '\0'**;**

fprintf**(**f**,** "%s\t->\t%s\n"**,** path**,** str**);**

**}**

void print\_fields**(**FILE **\***file**,** char **\*\***fields**)**

**{**

char buf**[**BUF\_SIZE**];**

const size\_t len **=** fread**(**buf**,** 1**,** BUF\_SIZE**,** file**);**

buf**[**len **-** 1**]** **=** '\0'**;**

char **\***value**,** **\*\***pfield**;**

**for** **(**

value **=** strtok**(**buf**,** " "**),** pfield **=** fields**;**

value **&&** **\***pfield**;** **++**pfield**)**

**{**

fprintf**(**f**,** "%-22s %s\n"**,** **\***pfield**,** value**);**

value **=** strtok**(NULL,** " "**);**

**}**

**}**

void print\_proc\_mem**(**FILE **\***file**)**

**{**

char buf**[**BUF\_SIZE**];**

int len**,** i**;**

**while** **((**len **=** fread**(**buf**,** 1**,** BUF\_SIZE**,** file**))** **>** 0**)**

**{**

**for** **(**i **=** 0**;** i **<** len**;** i**++)**

**if** **(**buf**[**i**]** **==** 0**)**

buf**[**i**]** **=** 10**;** // '\n'

buf**[**len**]** **=** 0**;**

fprintf**(**f**,** "%s"**,** buf**);**

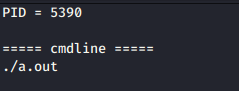
**}**

**}**

**Результат работы программы**

**1) CMDLINE**

На рисунке представлен вывод файла cmdline. Данный файл содержит полную командную строку процесса, если процесс не находится в состоянии зомби, иначе файл пуст.

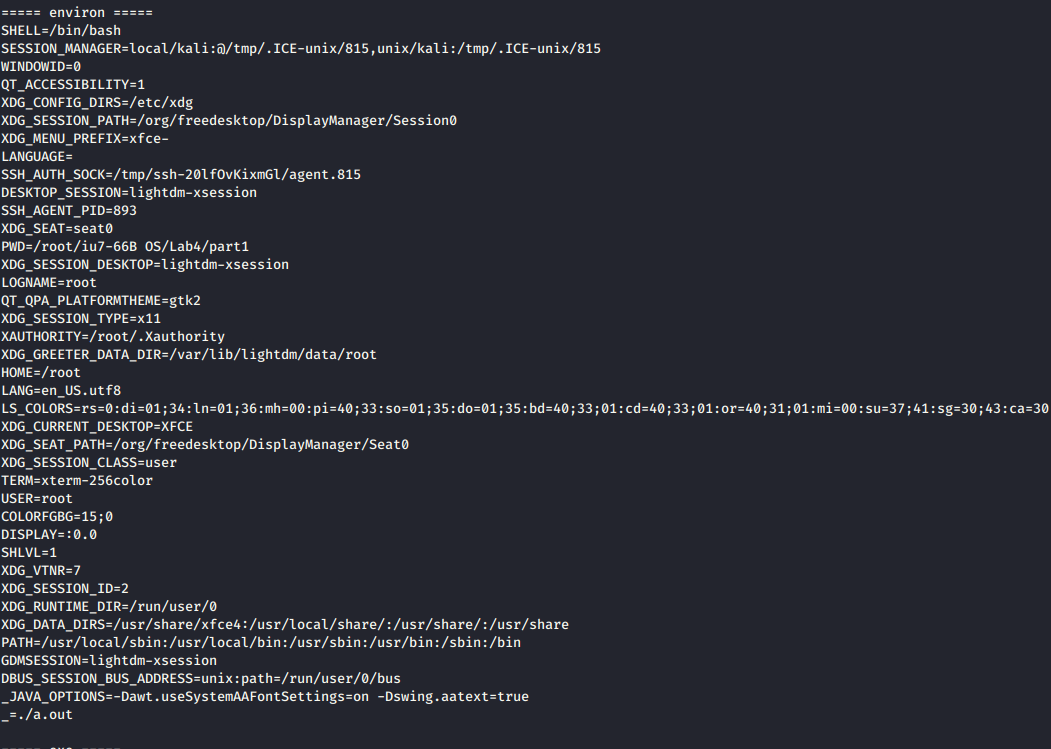
****

**2) cwd - символическая ссылка. Указывает на директорию процесса.**

****

**3) Содержание файла ENVIRON**

На рисунке представлено содержимое файла environ. Данный файл содержит исходное окружение, которое было установлено при запуске текущего процесса. Переменные окружения разделены символами конца строки (для доступа к поддиректории текущего '\0').

****

**4) EXE — представляет собой символическую ссылку, содержащий фактический путь к выполненной команде**



**5) Содержание директории FD**

На рисунке представлено содержимое директории fd. Данная поддиректория содержит одну запись для каждого файла, который открыт процессом. Имя каждой такой записи соответствует номеру файлового дескриптора и является символьной ссылкой на реальный файл.



**6) maps - файл, который содержит список выделенных участков памяти, используемых процессом и права доступа**

**Пример первой строки:**

1. 55e045cc6000 - начальный адрес выделенного участка памяти

2. 55e045cc8000 - конечный адрес выделенного участка памяти

3. r--p - права доступа

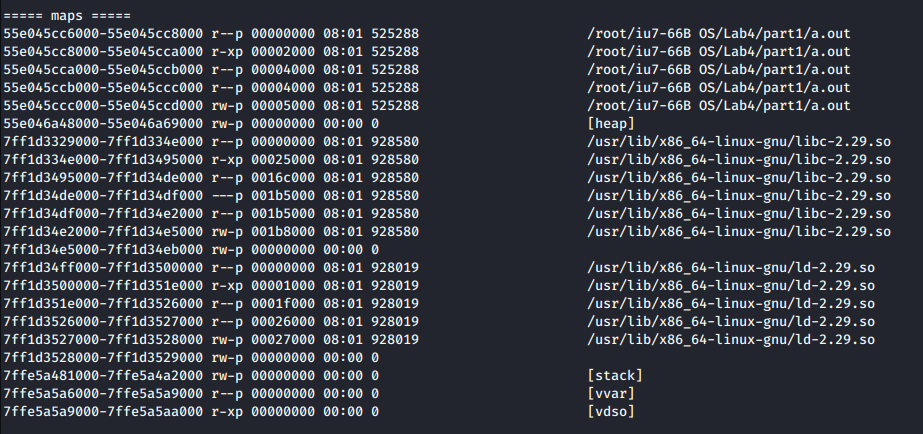
r - можно читать, w - можно писать, x - можно выполнять, s - shared разделяемый, p - private(copy on write)

4. 00000000 - смещение от которого выполняется распределение.

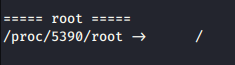
5. 08:01 - старший и младший номер устройства. (

6) 525288 - номер inode

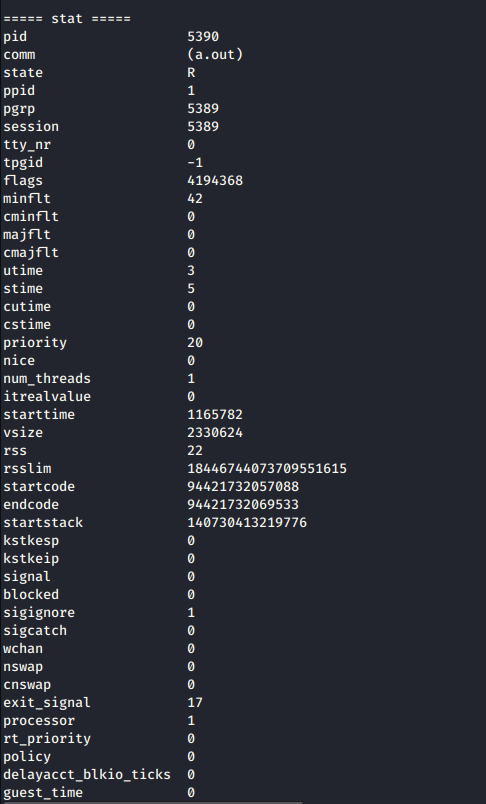
В системе имя файла не является идентификатором. Идентификатором является inode, соответственно inode на самом деле это структура, но есть в системе так называемые метаданные, это номера inode которые мы с вами видели когда набирали ls -ail.

****

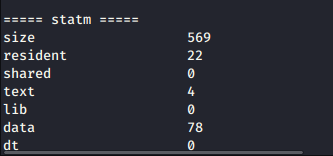
**7) root - это символическая ссылка(softlink). Указывает на корень файловой системы которой принадлежит образ процесса.**



**8) Stat – информация о состоянии процесса**

****

**9) statm - предоставляет информацию об использовании памяти, измеряемой в страницах**

****

**Часть 2.**

**Copy\_to\_user**

**Листинг**

**#include <linux/module.h>**

**#include <linux/kernel.h>**

**#include <linux/init.h>**

**#include <linux/vmalloc.h>**

**#include <linux/proc\_fs.h>**

**#include <linux/uaccess.h>**

**MODULE\_LICENSE("GPL");**

**MODULE\_AUTHOR("Dedich Borislav Borislavovich");**

**#define MAX\_BUF\_SIZE PAGE\_SIZE**

**#define DIRNAME "fortune\_dir"**

**#define FILENAME "fortune"**

**#define SYMLINK "fortune\_ln"**

**#define FILEPATH DIRNAME "/" FILENAME**

**static struct proc\_dir\_entry \*fortune\_dir = NULL;**

**static struct proc\_dir\_entry \*fortune = NULL;**

**static struct proc\_dir\_entry \*fortune\_ln = NULL;**

**static char \*cookie\_buffer;**

**static int next\_index;**

**static int current\_fortune;**

**static char tmp[MAX\_BUF\_SIZE];**

**ssize\_t fortune\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*offp)**

**{**

**int len;**

**printk("+: read called\n");**

**if (\*offp > 0 || !next\_index)**

**{**

**printk("+: offp case");**

**return 0;**

**}**

**if (current\_fortune >= next\_index)**

**{**

**current\_fortune = 0;**

**}**

**len = snprintf(tmp, MAX\_BUF\_SIZE, "%s\n", &cookie\_buffer[current\_fortune]);**

**if (copy\_to\_user(buf, tmp, len))**

**{**

**printk(KERN\_ERR "+: copy\_to\_user error\n");**

**return -EFAULT;**

**}**

**current\_fortune += len;**

**\*offp += len;**

**return len;**

**}**

**ssize\_t fortune\_write(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*offp)**

**{**

**if (len > MAX\_BUF\_SIZE - next\_index + 1)**

**{**

**printk(KERN\_ERR "+: cookie\_buffer overflow error\n");**

**return -ENOSPC;**

**}**

**if (copy\_from\_user(&cookie\_buffer[next\_index], buf, len))**

**{**

**printk(KERN\_ERR "+: copy\_to\_user error\n");**

**return -EFAULT;**

**}**

**next\_index += len;**

**cookie\_buffer[next\_index - 1] = '\0';**

**return len;**

**}**

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

**{**

**printk(KERN\_INFO "+: called open\n");**

**return 0;**

**}**

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

**{**

**printk(KERN\_INFO "+: called release\n");**

**return 0;**

**}**

**static struct file\_operations fops = {**

**.read = fortune\_read,**

**.write = fortune\_write,**

**.open = fortune\_open,**

**.release = fortune\_release};**

**static void freemem(void)**

**{**

**if (fortune\_ln)**

**{**

**remove\_proc\_entry(SYMLINK, NULL);**

**}**

**if (fortune)**

**{**

**remove\_proc\_entry(FILENAME, fortune\_dir);**

**}**

**if (fortune\_dir)**

**{**

**remove\_proc\_entry(DIRNAME, NULL);**

**}**

**if (cookie\_buffer)**

**{**

**vfree(cookie\_buffer);**

**}**

**}**

**static int \_\_init fortune\_init(void)**

**{**

**if (!(cookie\_buffer = vmalloc(MAX\_BUF\_SIZE)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during vmalloc\n");**

**return -ENOMEM;**

**}**

**memset(cookie\_buffer, 0, MAX\_BUF\_SIZE);**

**if (!(fortune\_dir = proc\_mkdir(DIRNAME, NULL)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during directory creation\n");**

**return -ENOMEM;**

**}**

**else if (!(fortune = proc\_create(FILENAME, 0666, fortune\_dir, &fops)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during file creation\n");**

**return -ENOMEM;**

**}**

**else if (!(fortune\_ln = proc\_symlink(SYMLINK, NULL, FILEPATH)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during symlink creation\n");**

**return -ENOMEM;**

**}**

**next\_index = 0;**

**current\_fortune = 0;**

**printk(KERN\_INFO "+: module loaded\n");**

**return 0;**

**}**

**static void \_\_exit fortune\_exit(void)**

**{**

**freemem();**

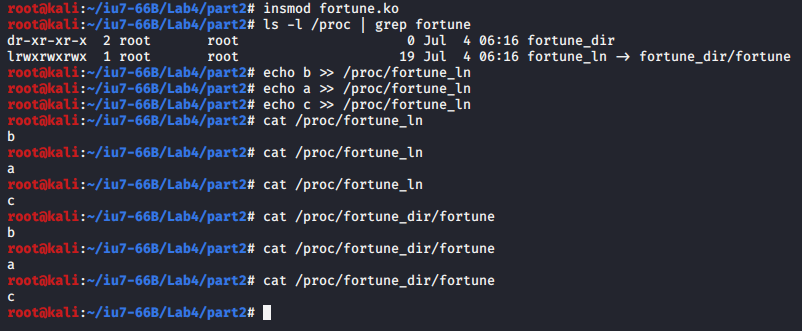
**printk(KERN\_INFO "+: module unloaded\n");**

**}**

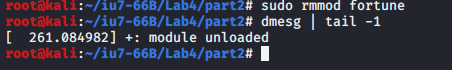
**module\_init(fortune\_init)**

**module\_exit(fortune\_exit)**

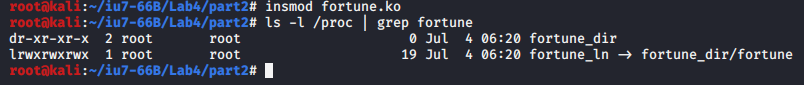
**Результат работы программы**



**Удаление**

****

**Новый файл и ссылка**



**Часть 2.**

# Листинг

|  |
| --- |
|  |

**#include <linux/module.h>**

**#include <linux/kernel.h>**

**#include <linux/init.h>**

**#include <linux/vmalloc.h>**

**#include <linux/proc\_fs.h>**

**#include <linux/uaccess.h>**

**#include <linux/seq\_file.h>**

**MODULE\_LICENSE("GPL");**

**MODULE\_AUTHOR("Dedich Borislav Borislavovich");**

**#define MAX\_BUF\_SIZE PAGE\_SIZE**

**#define DIRNAME "seqfile\_dir"**

**#define FILENAME "seqfile"**

**#define SYMLINK "seqfile\_ln"**

**#define FILEPATH DIRNAME "/" FILENAME**

**static struct proc\_dir\_entry \*fortune\_dir = NULL;**

**static struct proc\_dir\_entry \*fortune = NULL;**

**static struct proc\_dir\_entry \*fortune\_ln = NULL;**

**static char \*cookie\_buffer;**

**static int next\_index;**

**static int current\_fortune;**

**static char tmp[MAX\_BUF\_SIZE];**

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

**{**

**int len;**

**if (!next\_index)**

**{**

**return 0;**

**}**

**len = snprintf(tmp, MAX\_BUF\_SIZE, "%s", &cookie\_buffer[current\_fortune]);**

**seq\_printf(m, "%s", tmp);**

**current\_fortune += len;**

**printk(KERN\_INFO "+: show is called\n");**

**return 0;**

**}**

**ssize\_t fortune\_write(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*offp)**

**{**

**if (len > MAX\_BUF\_SIZE - next\_index + 1)**

**{**

**printk(KERN\_ERR "+: cookie\_buffer overflow error\n");**

**return -ENOSPC;**

**}**

**if (copy\_from\_user(&cookie\_buffer[next\_index], buf, len))**

**{**

**printk(KERN\_ERR "+: copy\_to\_user error\n");**

**return -EFAULT;**

**}**

**printk(KERN\_INFO "+: write is called\n");**

**next\_index += len - 1;**

**cookie\_buffer[next\_index] = '\0';**

**return len;**

**}**

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

**{**

**printk(KERN\_INFO "+: called open\n");**

**return single\_open(file, fortune\_show, NULL);**

**}**

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

**{**

**printk(KERN\_INFO "+: called release\n");**

**return single\_release(inode, file);**

**}**

**static struct file\_operations fops = {**

**.read = seq\_read,**

**.write = fortune\_write,**

**.open = fortune\_open,**

**.release = fortune\_release};**

**static void freemem(void)**

**{**

**if (fortune\_ln)**

**{**

**remove\_proc\_entry(SYMLINK, NULL);**

**}**

**if (fortune)**

**{**

**remove\_proc\_entry(FILENAME, fortune\_dir);**

**}**

**if (fortune\_dir)**

**{**

**remove\_proc\_entry(DIRNAME, NULL);**

**}**

**if (cookie\_buffer)**

**{**

**vfree(cookie\_buffer);**

**}**

**}**

**static int \_\_init fortune\_init(void)**

**{**

**if (!(cookie\_buffer = vmalloc(MAX\_BUF\_SIZE)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during vmalloc\n");**

**return -ENOMEM;**

**}**

**memset(cookie\_buffer, 0, MAX\_BUF\_SIZE);**

**if (!(fortune\_dir = proc\_mkdir(DIRNAME, NULL)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during directory creation\n");**

**return -ENOMEM;**

**}**

**else if (!(fortune = proc\_create(FILENAME, 0666, fortune\_dir, &fops)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during file creation\n");**

**return -ENOMEM;**

**}**

**else if (!(fortune\_ln = proc\_symlink(SYMLINK, NULL, FILEPATH)))**

**{**

**freemem();**

**printk(KERN\_ERR "+: error during symlink creation\n");**

**return -ENOMEM;**

**}**

**next\_index = 0;**

**current\_fortune = 0;**

**printk(KERN\_INFO "+: module loaded\n");**

**return 0;**

**}**

**static void \_\_exit fortune\_exit(void)**

**{**

**freemem();**

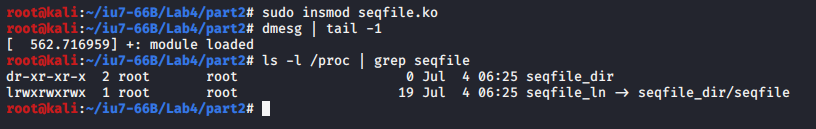
**printk(KERN\_INFO "+: module unloaded\n");**

**}**

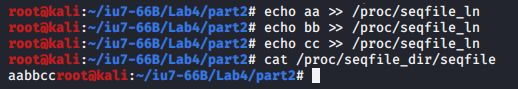
**module\_init(fortune\_init)**

**module\_exit(fortune\_exit)**

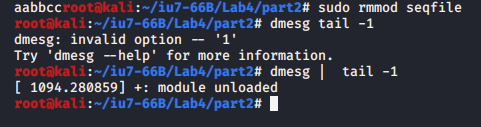
**Новый файл и ссылка**

****

**Пример работы програмы**

****

**Удаление**

****