|  |  |
| --- | --- |
|  | **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

Турсунов Ж.Р.

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 "/home/asus/home/projects/os-unix/6-semestr/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");**

**// if (already\_running())**

**// {**

**// syslog(LOG\_ERR, "демон уже запущен");**

**// exit(1);**

**// }**

**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]);**

**}**

**// getchar();**

**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);**

**// printf("%s", 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)**

**{**

**// int mem\_fd = open(name, O\_RDONLY);**

**// ptrace(PTRACE\_ATTACH, pid, NULL, NULL);**

**// waitpid(pid, NULL, 0);**

**// lseek(mem\_fd, offset, SEEK\_SET);**

**// read(mem\_fd, buf, \_SC\_PAGE\_SIZE);**

**// ptrace(PTRACE\_DETACH, pid, NULL, NULL);**

**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);**

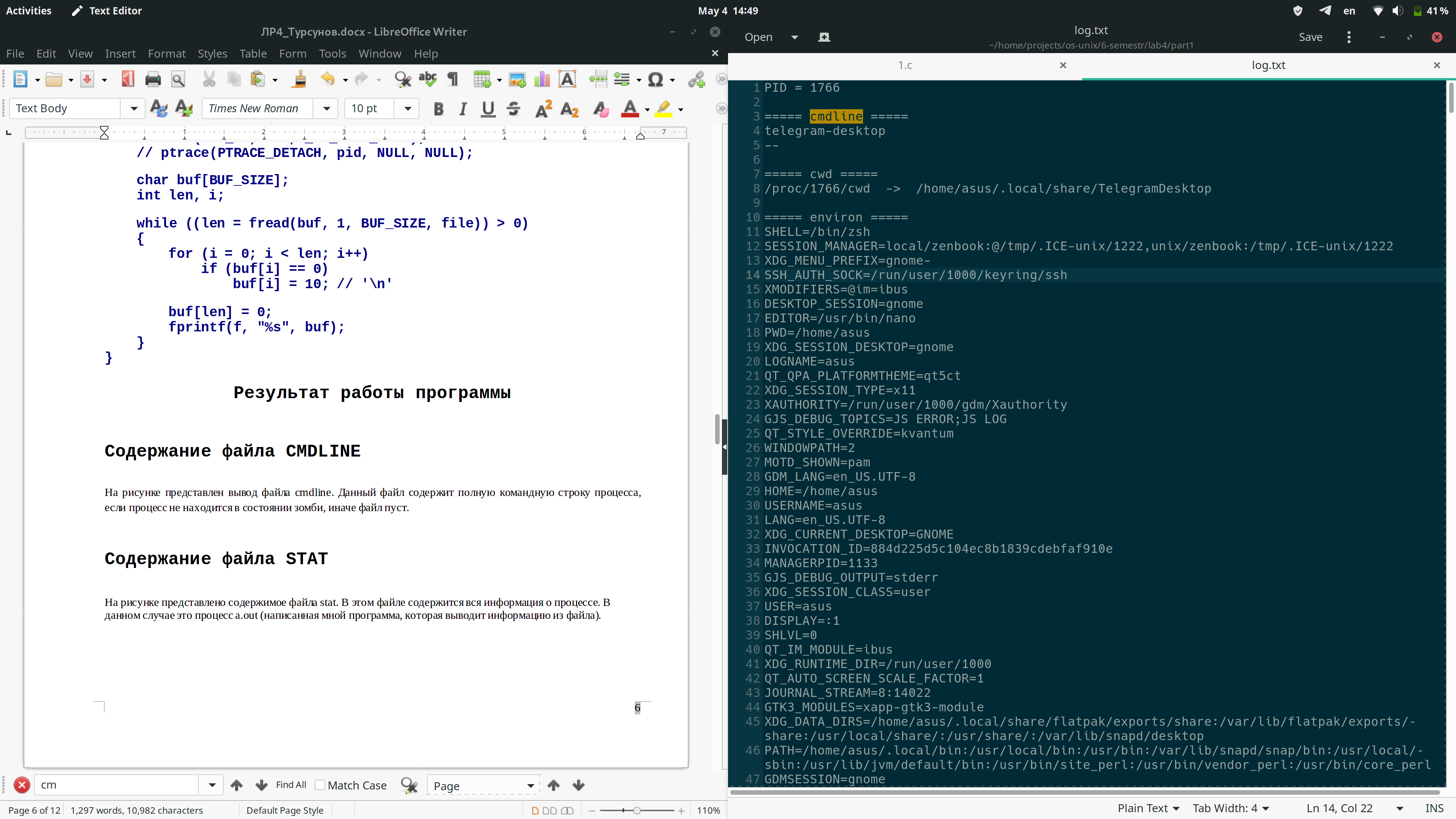
**}**

**}**

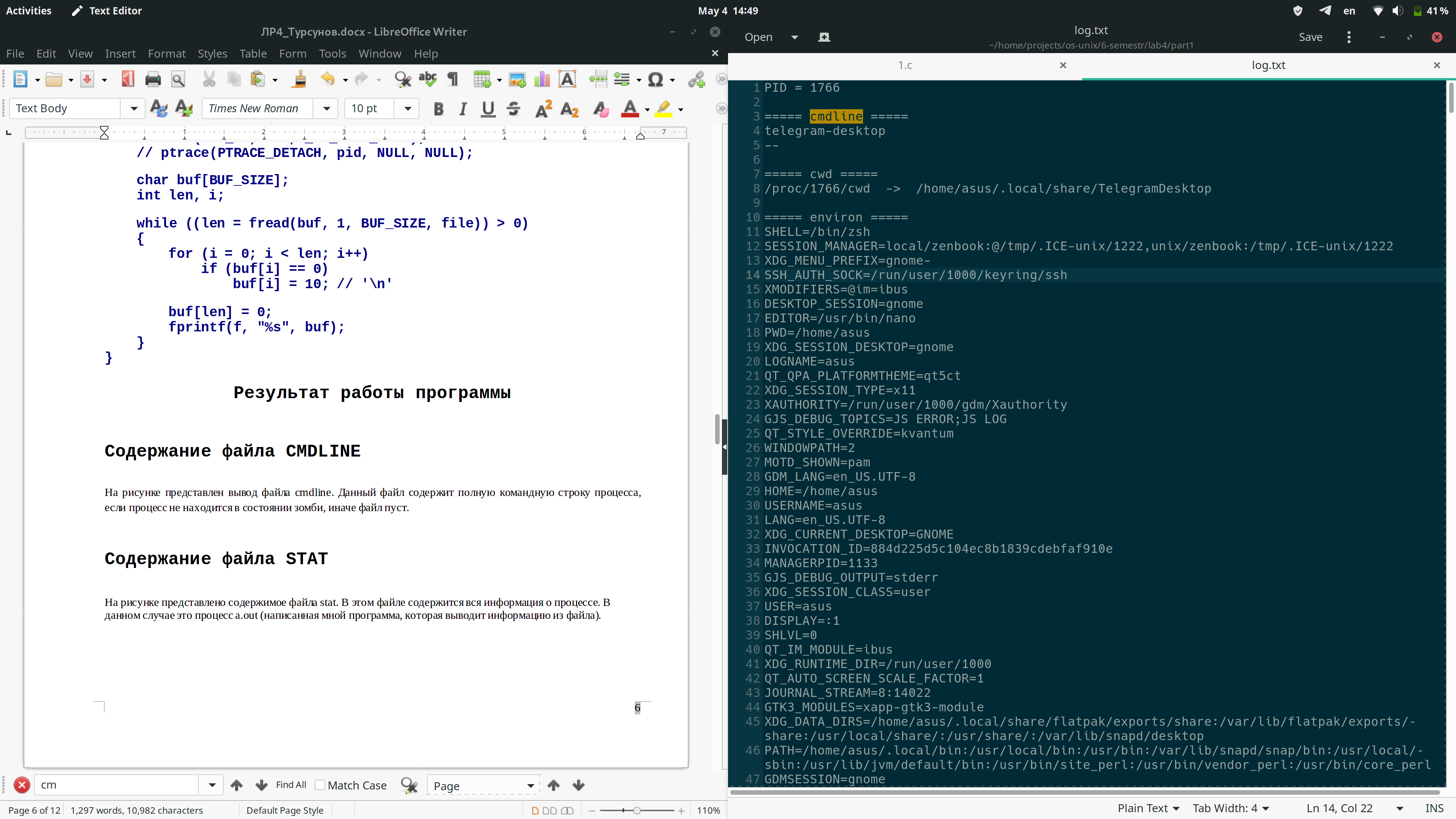
**Результат работы программы**

**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.

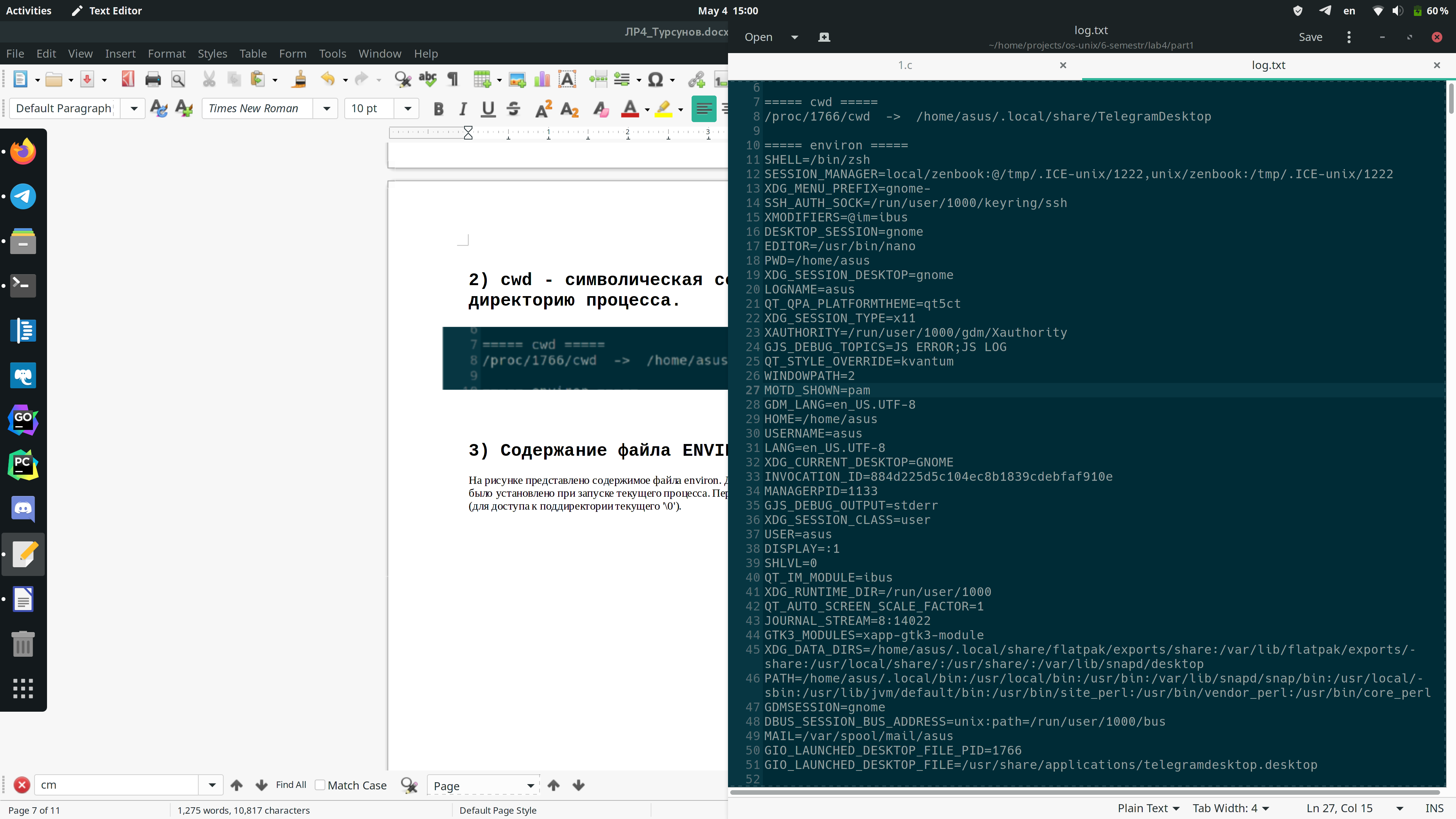


**2) cwd - символическая ссылка. Указывает на директорию процесса.**

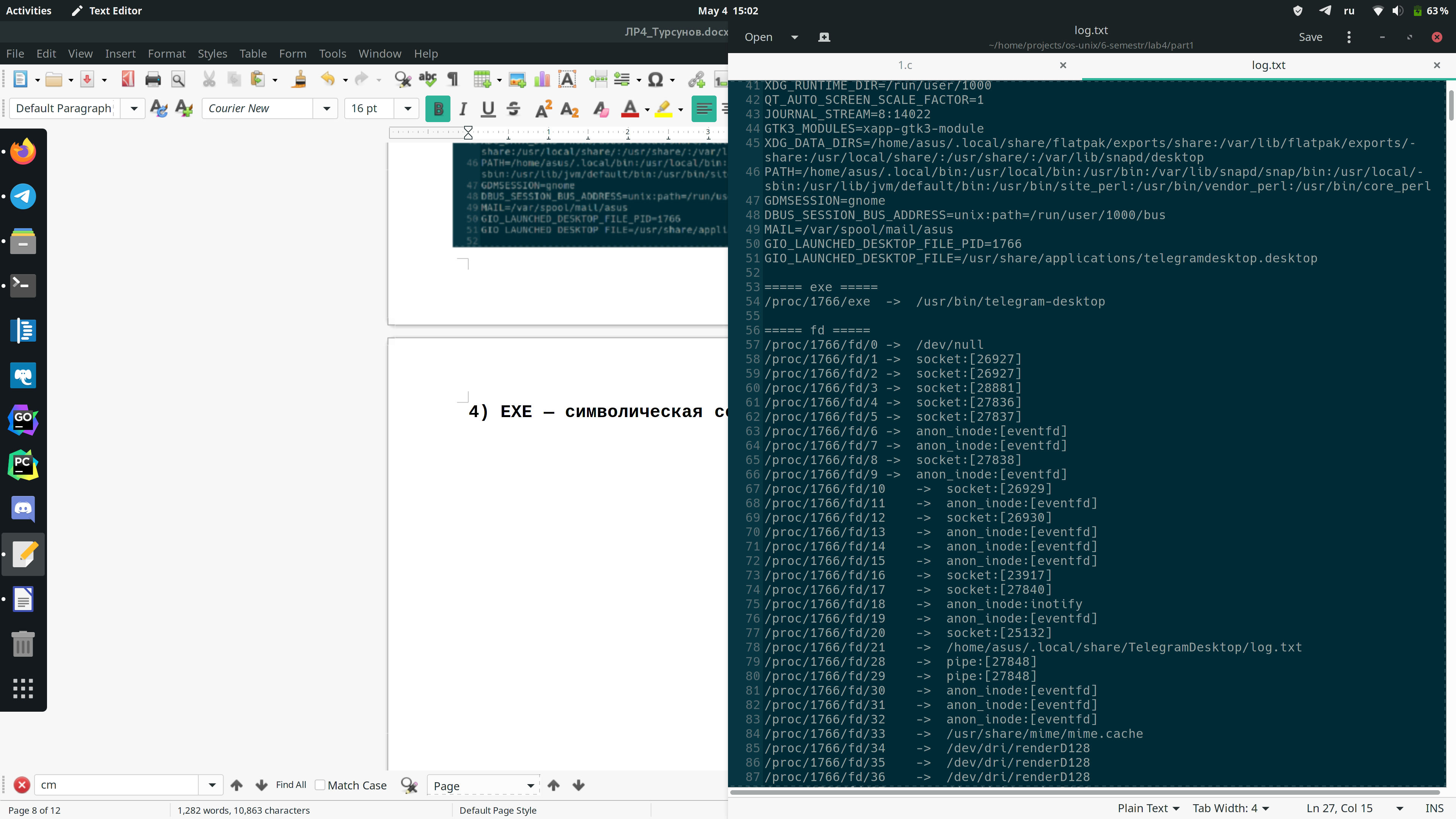


**3) Содержание файла ENVIRON**

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').

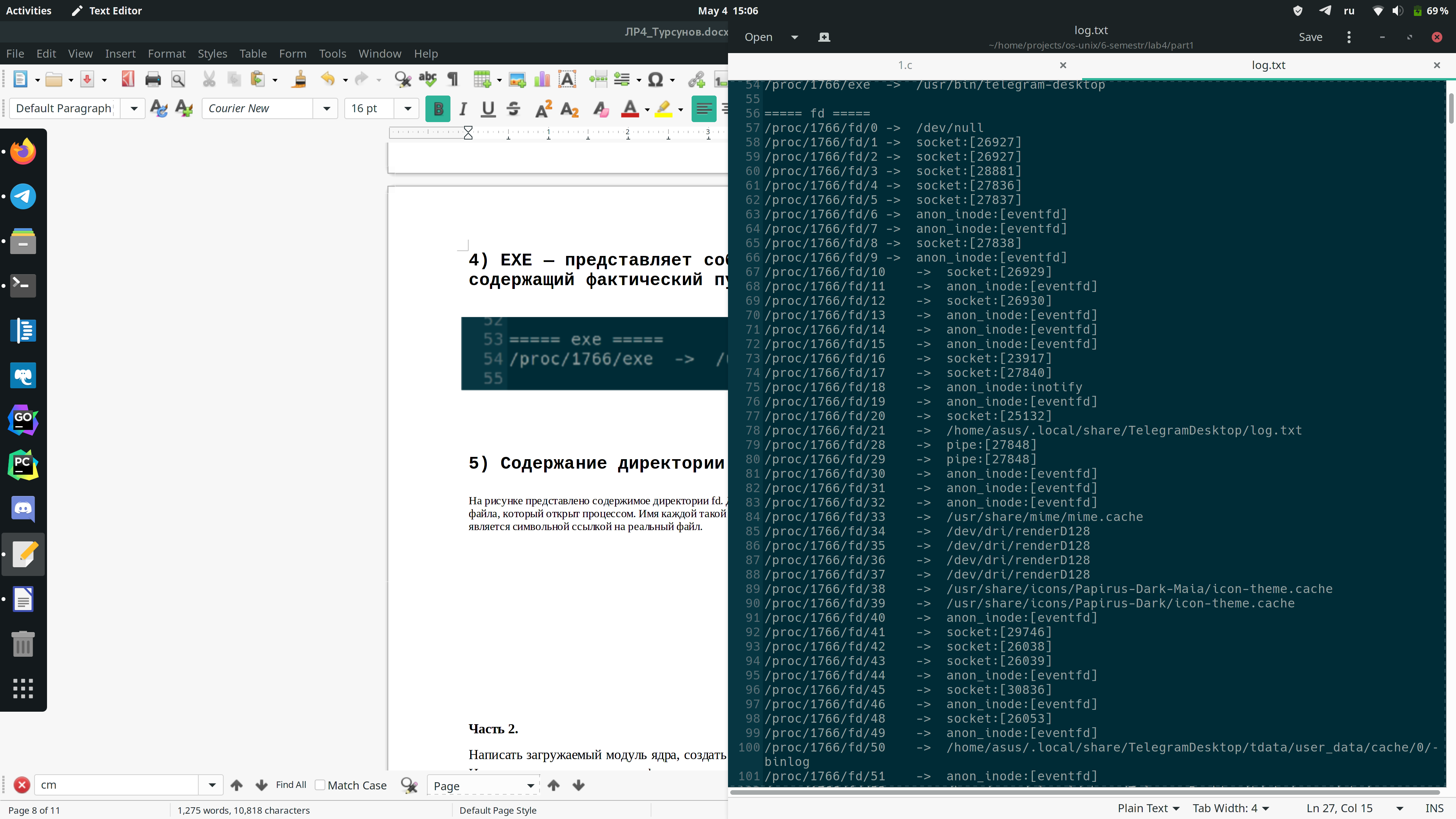
****

**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. 5620bb351000 - начальный адрес выделенного участка памяти

2. 5620bb730000 - конечный адрес выделенного участка памяти

3. r--p - права доступа

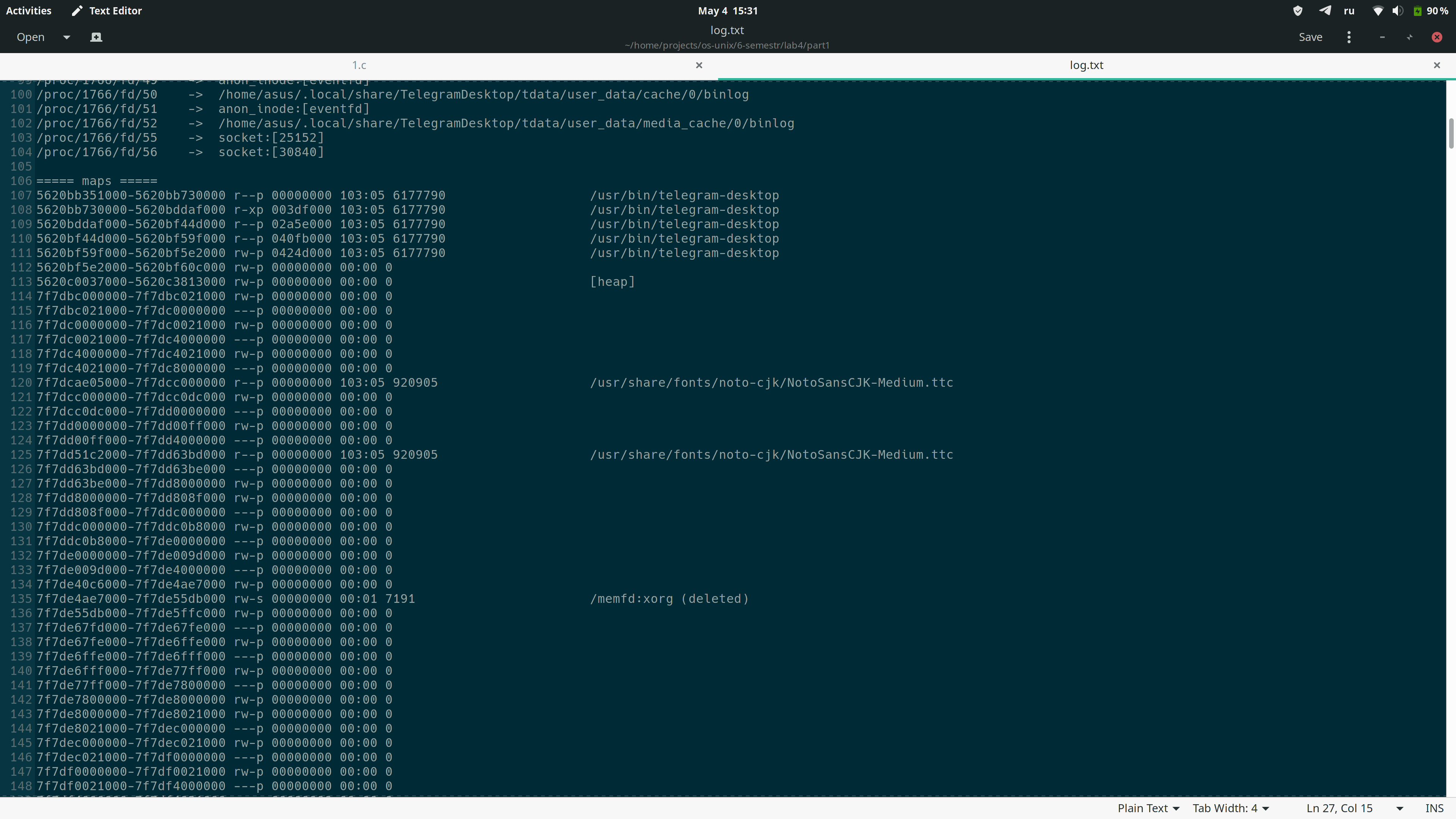
r - можно читать, w - можно писать, x - можно выполнять, s - shared разделяемый, p - private(copy on write)

4. 00000000 - смещение от которого выполняется распределение.

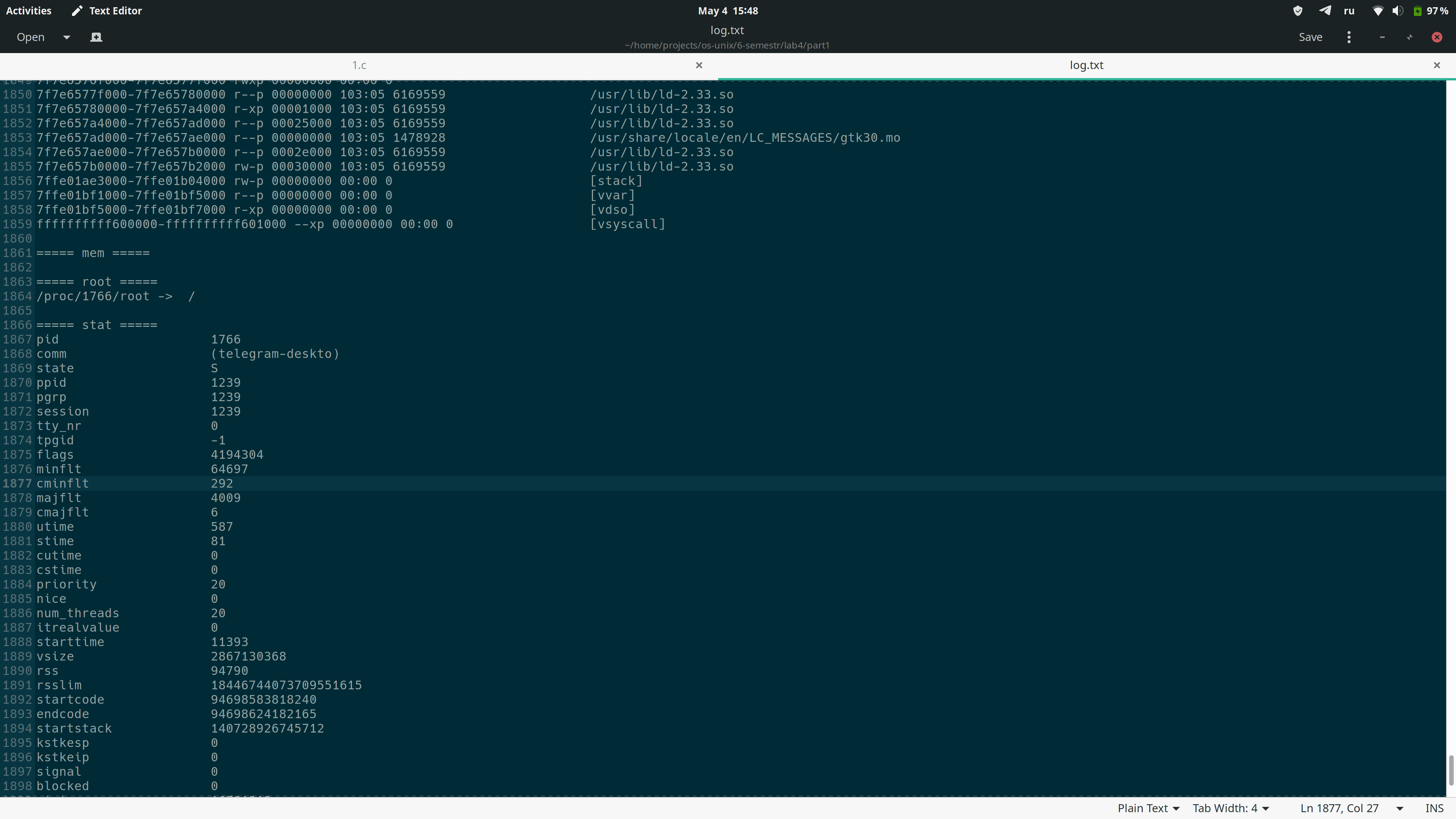
5. 103:05 - старший и младший номер устройства. (

6) 6177790 - номер inode

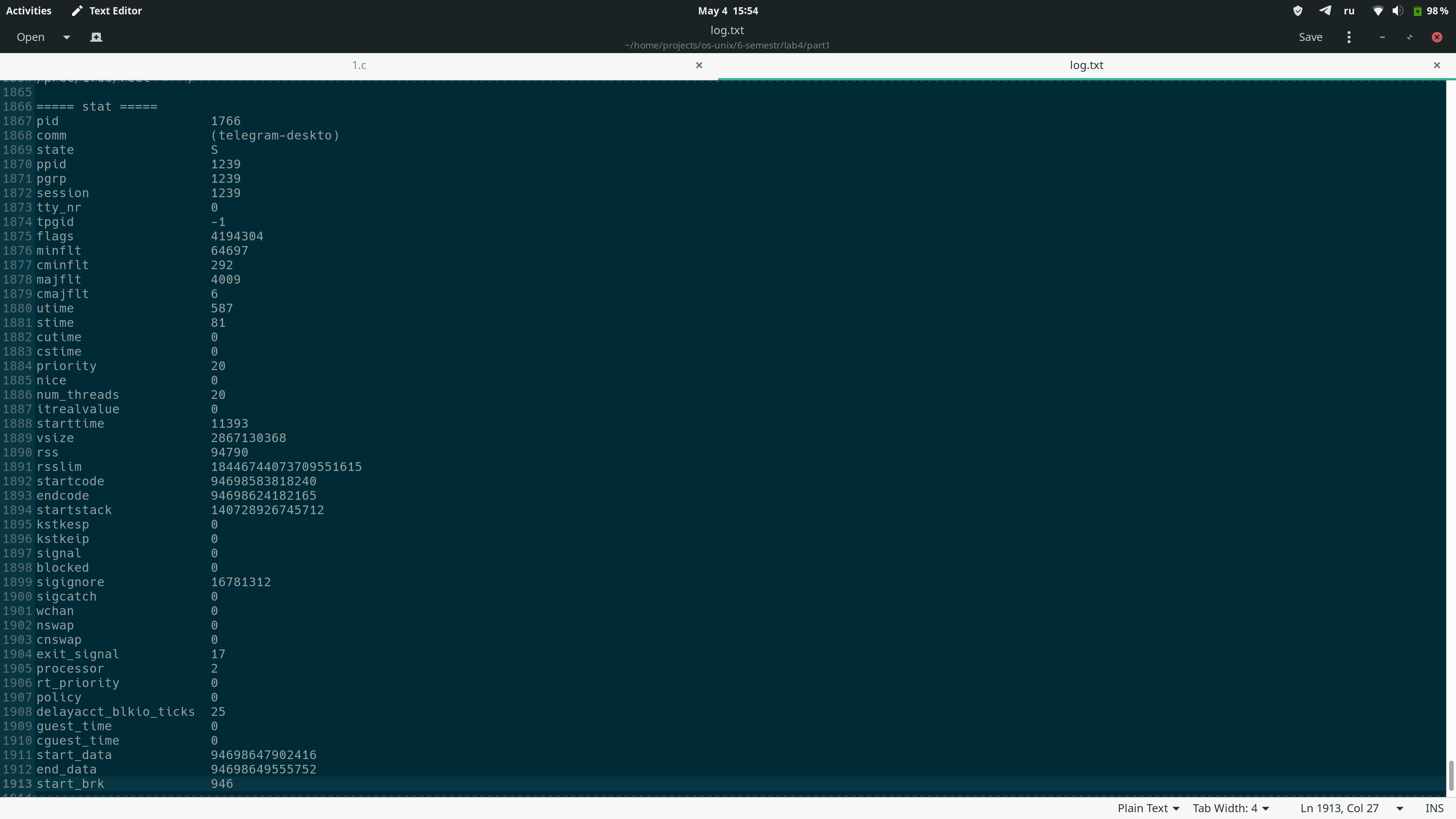
В системе имя файла не является идентификатором. Идентификатором является inode, соответственно inode на самом деле это структура, но есть в системе так называемые метаданные, это номера inode которые мы с вами видели когда набирали ls -ail.



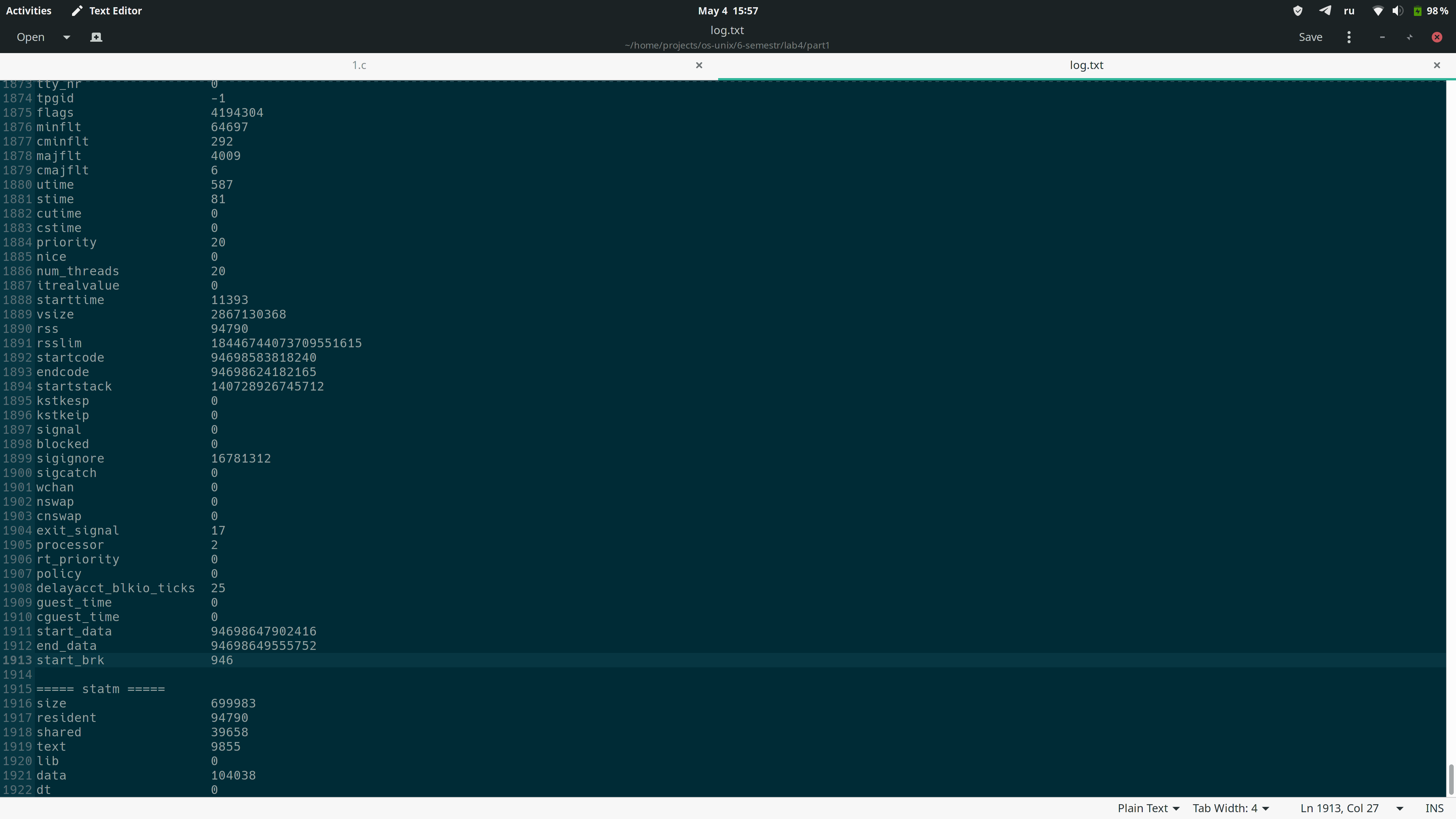
**7) root - это символическая ссылка(softlink). Указывает на корень файловой системы которой принадлежит образ процесса.**



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



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



**Часть 2.**

Написать загружаемый модуль ядра, создать файл в файловой системе proc, sysmlink, subdir. Используя соответствующие функции передать данные из пространства пользователя в пространство ядра (введенные данные вывести в файл ядра) и из пространства ядра в пространство пользователя. Продемонстрировать это.

# Листинг

|  |
| --- |
|  |

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

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

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

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

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

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

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

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

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

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

**static char \*str = NULL;**

**//индексы куда писать и откуда считывать**

**unsigned int write\_index;**

**unsigned int read\_index;**

**#define COOKIE\_POT\_SIZE PAGE\_SIZE**

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

**{**

**printk(KERN\_INFO "! Call my\_show\n");**

**//seq\_printf стандартная функция, выполняет действия, аналогичные copy\_to\_user или sprintf.**

**//int seq\_printf(struct seq\_file \*sfile, const char \*fmt, ...);**

**// Это эквивалент printf для реализаций seq\_file; он принимает обычную строку формата и дополнительные аргументы значений.**

**// Однако, вы также должны передать ей структуру seq\_file, которая передаётся в функцию show.**

**// Если seq\_printf возвращает ненулевое значение, это означает, что буфер заполнен и вывод будет отброшен. Большинство реализаций, однако, игнорирует возвращаемое значение.**

**seq\_printf(m, "Index is %u, message is %s\n", read\_index, str + read\_index);**

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

**if (len)**

**read\_index += len + 1;**

**return 0;**

**}**

**static ssize\_t my\_write(struct file\* file, const char \_\_user \*buffer, size\_t count, loff\_t \*f\_pos)**

**{**

**printk(KERN\_INFO "! Call my\_write\n");**

**if (copy\_from\_user(&str[write\_index], buffer, count)) // (куда, откуда, сколько байт)**

**return -EFAULT; //ошибка сегментирования (минус так как соглашение)**

**write\_index += count;**

**str[write\_index-1] = 0;**

**return count;**

**}**

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

**{**

**printk(KERN\_INFO "! Call my\_open\n");**

**// чтобы создать один файловый экземпляр модуля используется single\_open который передаёт адрес функции my\_show, а функция my\_show передаёт адрес страницы памяти**

**return single\_open(file, my\_show, NULL); // стандартная функция, может быть одновременно вызвана только одним процессом (чтобы открыть определенный файл) cat**

**}**

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

**{**

**printk(KERN\_INFO "! Сall my\_release\n");**

**return single\_release(inode, file); //выгружает seq file**

**}**

**static struct file\_operations fortune\_proc\_ops={**

**//.proc\_owner = THIS\_MODULE,**

**.open = my\_open,**

**.release = my\_release,**

**.read = seq\_read,**

**//.llseek = seq\_lseek,**

**.write = my\_write**

**};**

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

**{**

**printk(KERN\_INFO "! Call fortune init\n");**

**write\_index = 0;**

**read\_index = 0;**

**// выделить память для строки**

**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); //заполняем строку нулями**

**//Чтобы работать с виртуальной файловой системой proc в ядре, в ядре определена структура**

**struct proc\_dir\_entry \*entry;**

**entry = proc\_create("fortune", S\_IRUGO | S\_IWUGO, NULL, &fortune\_proc\_ops); //создаёт файл в виртуальной системе проц - имя файла, права доступа, указатель на родителя (если Null то создастся в корне), указатель на операции**

**if(!entry)**

**{**

**vfree(str);**

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

**return -ENOMEM;cd 02-evdev**

**}**

**// создать каталог в файловой системе /proc**

**proc\_mkdir("cookie\_dir", NULL);**

**// создать символическую ссылку на "/proc/fortune"**

**proc\_symlink("cookie\_symlink", NULL, "/proc/fortune");**

**printk(KERN\_INFO "Fortune module loaded successfully\n");**

**return 0;**

**}**

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

**{**

**printk(KERN\_INFO "! Сall fortune exit\n");**

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

**remove\_proc\_entry("cookie\_dir", NULL); //Чтобы работать с виртуальной файловой системой proc в ядре, в ядре определена структура**

**remove\_proc\_entry("cookie\_symlink", NULL);**

**if (str)**

**vfree(str);**

**printk(KERN\_INFO "Fortune module unloaded\n"); //dmesq**

**}**

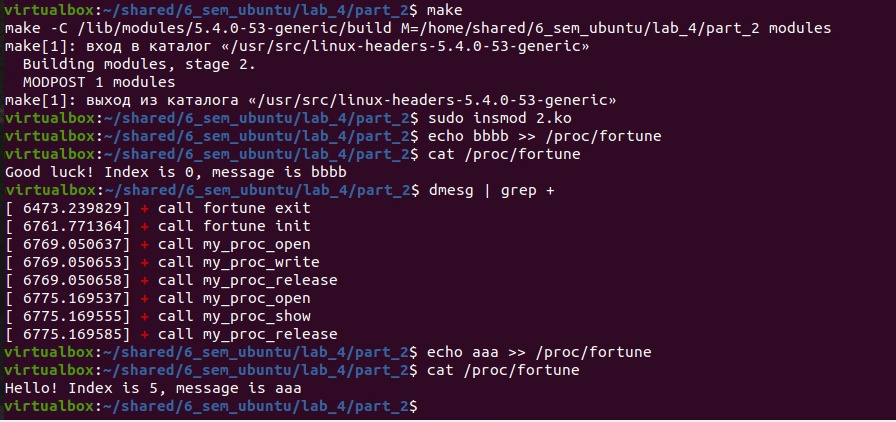
**module\_init(fortune\_init);**

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

**// для release и open нужен inode так как там надо работать открывать закрывать файлы, а read и write работают уже с открытыми.**

**//dmesq -C очистить журнал**

**Результат работы программы**

****