



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический  
университет имени Н.Э. Баумана»  
(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	Сиденко А.Г.
Группа	ИУ7-63Б
Оценка (баллы)	
Преподаватель	Рязанова Н.Ю.

Москва, 2020 г.

## Часть 1

**Задание:** Используя виртуальную файловую систему `proc` вывести информацию об окружении процесса, информацию, характеризующую состояние процесса, содержание директории `fd` и файла `cmdline`.

Листинг 1: Часть 1.

```
1 #include <sys/stat.h>
2 #include <dirent.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <errno.h>
7 #include <unistd.h>
8
9 #define BUF_SIZE 0x1000
10
11 static char* param_stat[] = {"pid", "comm", "state", "ppid", "pgrp",
12 "session", "tty_nr", "tpgid", "flags", "minflt", "cminflt", "majflt",
13 "cmajflt", "utime", "stime", "cutime", "cstime", "priority", "nice",
14 "num_threads", "itrealvalue", "starttime", "vsize", "rss", "rsslim",
15 "startcode", "endcode", "startstack", "kstkesp", "kstkeip", "signal",
16 "blocked", "sigignore", "sigcatch", "wchan", "nswap", "cnsnap",
17 "exit_signal", "processor", "rt_priority", "policy",
18 "delayacct_blkio_ticks", "guest_time", "cguest_time", "start_data",
19 "end_data", "start_brk", "arg_start", "arg_end", "env_start",
20 "env_end", "exit_code"};
21
22 // Вывод содержимого файла ENVIRON
23 void readEnviron()
24 {
25     char buf[BUF_SIZE];
26     int len, i;
27     FILE *f = fopen("/proc/self/environ", "r");
28     if (f == NULL)
29     {
30         printf("%s", strerror(errno));
31         exit(errno);
32     }
33
34     while ((len = fread(buf, 1, BUF_SIZE, f)) > 0)
35     {
36         for (i = 0; i < len; i++)
37             if (buf[i] == 0)
38                 buf[i] = 10; // код 10, перевод строки
39         buf[len] = 0;
40         printf("%s", buf);
41     }
42
43     fclose(f);
```

```

44 }
45
46 // Вывод содержимого файла STAT, CMDLINE
47 void readFile(char *path)
48 {
49     char buf[BUF_SIZE];
50     FILE *f = fopen(path, "r");
51     if (f == NULL)
52     {
53         printf("%s", strerror(errno));
54         exit(errno);
55     }
56
57     fread(buf, 1, BUF_SIZE, f);
58     char *pch = strtok(buf, "_"); // поиск разделителей в файле
59
60     int i = 0;
61     while (pch != NULL)
62     {
63         if (strcmp("/proc/self/stat", path) == 0)
64             printf("%s=", param_stat[i]);
65         i++;
66         printf("%s\n", pch);
67         pch = strtok(NULL, "_"); // продолжение сканирование с того места,
68                                 // где был остановлен предыдущий
69                                 // успешный вызов функции
70     }
71
72     fclose(f);
73 }
74
75 // Вывод содержимого директории FD
76 void contentDir()
77 {
78     struct dirent *dirp;
79     DIR *dp;
80     char str[BUF_SIZE];
81     char path[BUF_SIZE];
82
83     if ((dp = opendir("/proc/self/fd/")) == NULL)
84     {
85         printf("%s", strerror(errno));
86         exit(errno);
87     }
88
89     while ((dirp = readdir(dp)) != NULL)
90     {
91         // Пропуск каталогов . и ..
92         if ((strcmp(dirp->d_name, ".")!=0)&&(strcmp(dirp->d_name, "..")!=0))
93         {
94             sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
95             readlink(path, str, BUF_SIZE); // Считывает значение

```

```

96 // символной ссылке
97     printf("%s->%s\n", dirp->d_name, str);
98 }
99 }
100
101 if (closedir(dp) < 0)
102 {
103     printf("%s", strerror(errno));
104     exit(errno);
105 }
106 }
107
108 int main(int argc, char *argv[])
109 {
110     if (argc != 2)
111     {
112         printf("Использование: ./output.exe <stat/environ/fd/cmdline>\n");
113         return 1;
114     }
115
116     if (strcmp("environ", argv[1]) == 0)
117         readEnviron();
118     else if (strcmp("stat", argv[1]) == 0)
119         readFile("/proc/self/stat");
120     else if (strcmp("fd", argv[1]) == 0)
121         contentDir("/proc/self/fd/");
122     else if (strcmp("cmdline", argv[1]) == 0)
123         readFile("/proc/self/cmdline");
124     else
125         printf("Использование: ./output.exe <stat/environ/fd/cmdline>\n");
126
127     return 0;
128 }

```

1. Список окружения процесса (environ):

```

lab4 @ ./output.exe environ
LANG=en_US.UTF-8
DISPLAY=:1
XDG_VTNR=2
LOGNAME=parallels
PWD=/home/parallels/Desktop/Parallels Shared Folders/Home/Desktop/university/3_course/sem6/Operating_systems/semestr2/lab4
XAUTHORITY=/run/user/1000/gdm/Xauthority
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
QT_QPA_PLATFORMTHEME=qqgnomeplatform
JOURNAL_STREAM=8:20516
COLORTERM=truecolor
XDG_SESSION_ID=4
DESKTOP_SESSION=default
XDG_SESSION_DESKTOP=default
GDMSESSION=default
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
USERNAME=parallels
WINDOWPATH=2
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
VTE_VERSION=4601
XDG_DATA_DIRS=/usr/share/gnome:/usr/local/share/:/usr/share/
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
XDG_MENU_PREFIX=gnome-
QT_ACCESSIBILITY=1
GDM_LANG=en_US.UTF-8
GJS_DEBUG_OUTPUT=stderr
XDG_SESSION_TYPE=x11
SHELL=/bin/zsh
WINDOWID=27262981
TERM=xterm-256color
GTK_MODULES=gail:atk-bridge
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XDG_CURRENT_DESKTOP=GNOME
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/home/parallels/.dotnet/tools
SSH_AGENT_PID=1738
HOME=/home/parallels
XDG_SEAT=seat0
XDG_RUNTIME_DIR=/run/user/1000
SESSION_MANAGER=local/debian-gnu-linux-vm:@/tmp/.ICE-unix/1641,unix/debian-gnu-linux-vm:/tmp/.ICE-unix/1641
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
USER=parallels

```

## 2. Информация о процессе (stat):

```

lab4 @ ./output.exe stat
pid=21441
comm=(output.exe)
state=R
ppid=5751
pgrp=21441
session=5751
tty_nr=34816
tpgid=21441
flags=4194304
minflt=90
cminflt=0
majflt=0
cmajflt=0
utime=0
stime=0
cutime=0
cstime=0
priority=20
nice=0
num_threads=1
itréalvalue=0
starttime=1409513
vsize=4263936
rss=158
rsslim=18446744073709551615
startcode=94574668247040
endcode=94574668253380
startstack=140727918495040
kstkesp=0
kstkeip=0
signal=0
blocked=0
sigignore=0
sigcatch=0
wchan=0
nswap=0
cnswap=0
exit_signal=17
processor=1
rt_priority=0
policy=0

```

3. Директория процесса (cmdline):

```
lab4 E ./output.exe cmdline  
./output.exe
```

4. Ссылки на файлы, которые «открыл» процесс (fd):

```
lab4 E ./output.exe fd  
0 -> /dev/pts/0  
1 -> /dev/pts/0  
2 -> /dev/pts/0  
3 -> /proc/32234/fd
```

## Часть 2

**Задание:** Написать загружаемый модуль ядра, создать файл в файловой системе `proc`, `symlink`, `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 2: Часть 2.

```
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/proc_fs.h>
4 #include <linux/string.h>
5 #include <linux/vmalloc.h>
6 #include <linux/fs.h>
7 #include <asm/uaccess.h>
8
9 MODULE_LICENSE("GPL");
10 MODULE_AUTHOR("Sidenko");
11 MODULE_DESCRIPTION("Lab4");
12
13 static struct proc_dir_entry *proc_entry, *dir, *symlink;
14 static char* buffer; // Хранилище 'фортунок'
15 int buffer_index, next_fortune; // Индексы для записи и вывода
16
17 ssize_t fortune_write(struct file *filp, const char __user *buf,
18                      size_t len, loff_t *offp)
19 {
20     int space_available = (PAGE_SIZE - buffer_index) + 1;
21
22     // Хватит ли места для размещения
23     if (len > space_available)
24         return -ENOSPC;
25
26     // Копирование строки
27     if (copy_from_user(&buffer[buffer_index], buf, len))
28         return -EFAULT;
29
30     buffer_index += len;
31     buffer[buffer_index - 1] = '\n';
32
33     return len;
34 }
35
36 ssize_t fortune_read(struct file *filp, char __user *buf, size_t count,
37                    loff_t *offp)
38 {
39     int len;
40     if (*offp > 0)
41         return 0;
```

```

42
43 // Перевод индекса на первый элемент
44 if (next_fortune >= buffer_index)
45     next_fortune = 0;
46
47 len = copy_to_user(buf, &buffer[next_fortune], count);
48 next_fortune += len;
49 *offp += len;
50
51 return len;
52 }
53
54 struct file_operations fileops =
55 {
56     .read = fortune_read,
57     .write = fortune_write
58 };
59
60 int fortune_module_init(void)
61 {
62     buffer = (char *)vmalloc(PAGE_SIZE);
63     if (!buffer)
64     {
65         printk(KERN_INFO "fortune:_No_memory_for_create_buffer\n");
66         return -ENOMEM;
67     }
68     memset(buffer, 0, PAGE_SIZE);
69
70     proc_entry = proc_create("fortune", 0666, NULL, &fileops);
71     if (proc_entry == NULL)
72     {
73         vfree(buffer);
74         printk(KERN_INFO "fortune:_Couldn't_create_proc_entry\n");
75         return -ENOMEM;
76     }
77
78     dir = proc_mkdir("fortune_dir", NULL);
79     symlink = proc_symlink("fortune_symlink", NULL, "/proc/fortune_dir");
80     if ((dir == NULL) || (symlink == NULL))
81     {
82         vfree(buffer);
83         printk(KERN_INFO "fortune:_Couldn't_create_proc_dir,_symlink\n");
84         return -ENOMEM;
85     }
86
87     buffer_index = 0;
88     next_fortune = 0;
89
90     printk(KERN_INFO "fortune:_Module_loaded.\n");
91     return 0;
92 }
93

```



```

94 void fortune_module_exit(void)
95 {
96     remove_proc_entry("fortune", NULL);
97     remove_proc_entry("fortune_symlink", NULL);
98     remove_proc_entry("fortune_dir", NULL);
99     vfree(buffer);
100     printk(KERN_INFO "fortune:_Module_unloaded.\n");
101 }
102
103 module_init(fortune_module_init);
104 module_exit(fortune_module_exit);

```

1. Собираем, загружаем модуль ядра, проверяем создание файла, директории, символической ссылки:

```

+ ~/Desktop/lab4 # make
make -C /lib/modules/4.9.0-8-amd64/build M=/home/parallels/Desktop/lab4 modules
make[1]: Entering directory '/usr/src/linux-headers-4.9.0-8-amd64'
  CC [M] /home/parallels/Desktop/lab4/fortune.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC /home/parallels/Desktop/lab4/fortune.mod.o
  LD [M] /home/parallels/Desktop/lab4/fortune.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.0-8-amd64'
+ ~/Desktop/lab4 # sudo insmod fortune.ko
+ ~/Desktop/lab4 # sudo dmesg | tail -1
[ 4140.380313] fortune: Module loaded.
+ ~/Desktop/lab4 # ls -l /proc | grep fortune
-rw-rw-rw- 1 root      root      0 Mar 21 16:47 fortune
dr-xr-xr-x 2 root      root      0 Mar 21 16:47 fortune_dir
lrwxrwxrwx 1 root      root     17 Mar 21 16:47 fortune_symlink -> /proc/fortune_dir

```

2. Посылка и считывание данных:

```

+ ~/Desktop/lab4 # echo "Hi" >> /proc/fortune
+ ~/Desktop/lab4 # echo "How are you?" >> /proc/fortune
+ ~/Desktop/lab4 # cat /proc/fortune
Hi
How are you?

```

3. Выгружаем модуль:

```

+ ~/Desktop/lab4 # sudo rmmod fortune
+ ~/Desktop/lab4 # sudo dmesg | tail -2
[ 4140.380313] fortune: Module loaded.
[ 4224.255873] fortune: Module unloaded.

```

## Почему для передачи данных в ядро и из ядра в режим пользователя нужны специальные функции?

В Linux память сегментирована, указатель ссылается не на уникальную позицию в памяти, а ссылается на позицию в сегменте.

Фактически само ядро и каждый из процессов располагаются в своих собственных изолированных адресных пространствах.

При выполнении обычной программы адресация происходит автоматически в соответствии с принятой в системе.

Соответственно, если выполняется код ядра и необходимо получить доступ к сегменту кода ядра, то нужен буфер. Но когда хотим передать информацию между процессом, запущенным в режиме пользователя, и ядром, то соответствующая функция ядра получит указатель на буфер процесса.