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 Операционые системы.
Тема Вирутальная файловая система / proc

Студент Куприй А. А.
Группа ИУ7-63Б
Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

1 Практическая часть

1.1 Задание №1

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

Листинг 1.1 — Вывод содержимого environ

```
1  void print_environ()
2  {
3    char buf[BUF_SIZE] = {0};
4    int len, i;
5    FILE *fd = fopen("/proc/self/environ", "r");
6    if (fd == NULL)
7    {
8      printf("%s", strerror(errno));
9      exit(errno);
10   }
11
12   while ((len = fread(buf, 1, BUF_SIZE, fd)) > 0)
13   {
14     for (i = 0; i < len; i++)
15       if (buf[i] == 0)
16       {
17         buf[i] = 10;
18       }
19     buf[len] = 0;
20     printf("%s", buf);
21   }
22
23   fclose(fd);
24 }
```

Листинг 1.2 — Вывод содержимого файла для stat и cmdline

```
1  void print_file(char *path)
2  {
3      char buf[BUF_SIZE] = {0};
4      char *pch;
5      int i = 0;
6      FILE *f = fopen(path, "r");
7      if (f == NULL)
8      {
9          printf("%s", strerror(errno));
10         exit(errno);
```

```
11      }
12
13      fread(buf, 1, BUF_SIZE, f);
14      pch = strtok(buf, " ");
15
16      while (pch)
17      {
18          if (strcmp("/proc/self/stat", path) == 0)
19          {
20              printf("%s=", param_stat[i++]);
21          }
22          printf("%s\n", pch);
23          pch = strtok(NULL, " ");
24      }
25
26      fclose(f);
27  }
```

Листинг 1.3 — Вывод содержимого директории

```
1  void print_fd()
2  {
3    struct dirent *dirp;
4    DIR *dp;
5    char path[BUF_SIZE];
6    char str[BUF_SIZE];
7
8    if (!(dp = opendir("/proc/self/fd/")))
9      {
10     printf("%s", strerror(errno));
11     exit(errno);
12   }
13
14   while ((dirp = readdir(dp)))
15   {
16     if ((strcmp(dirp->d_name, ".") != 0) && (strcmp(dirp->d_name, "..") !=
         0))
17     {
18       sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
19       readlink(path, str, BUF_SIZE);
20       printf("%s -> %s\n", dirp->d_name, str);
21     }
22   }
23
24   if (closedir(dp) < 0)
25   {
26     printf("%s", strerror(errno));
```

```
27          exit(errno);
28      }
29  }
```

Результат:

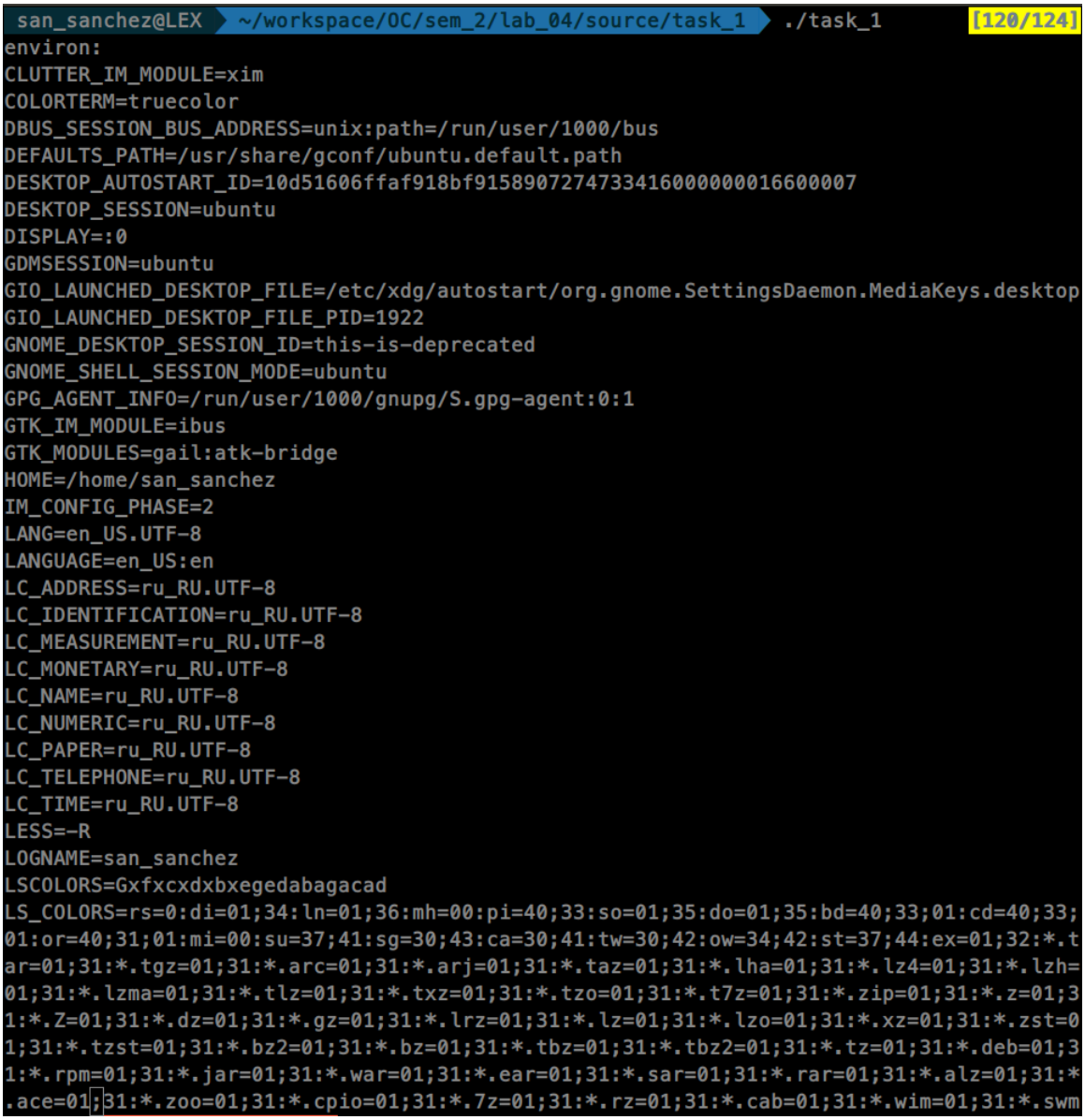

Рисунок 1.1 — Скришот вывода информации об окружении процесса (часть 1).

```
PATH=/home/san_sanchez/workspace/datagrip-2019.2.5/DataGrip-2019.2.5/bin:/opt/m[66/124]
ls/bin:/home/san_sanchez/bin:/usr/local/bin:/home/san_sanchez/workspace/datagrip-2019.2
.5/DataGrip-2019.2.5/bin:/opt/mssql-tools/bin:/home/san_sanchez/bin:/usr/local/bin:/hom
e/san_sanchez/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:/home/san_sanchez/.fzf/bin
PWD=/home/san_sanchez/workspace/OC/sem_2/lab_04/source/task_1
QT4_IM_MODULE=xim
QT_ACCESSIBILITY=1
QT_IM_MODULE=ibus
SESSION_MANAGER=local/LEX:@/tmp/.ICE-unix/1660,unix/LEX:/tmp/.ICE-unix/1660
SHELL=/usr/bin/zsh
SHLVL=2
SSH_AGENT_PID=1738
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
S_COLORS=auto
TERM=screen-256color
TERMINATOR_DBUS_NAME=net.tenshu.Terminator20x1a6021154d881c
TERMINATOR_DBUS_PATH=/net/tenshu/Terminator2
TERMINATOR_UUID=urn:uuid:a8abd3aa-434c-4736-bf95-bb431352207d
TEXTDOMAIN=im-config
TEXTDOMAINDIR=/usr/share/locale/
TMUX=/tmp/tmux-1000/default,2175,0
TMUX_PANE=%1
USER=san_sanchez
USERNAME=san_sanchez
VTE_VERSION=5202
WINDOWPATH=2
XAUTHORITY=/run/user/1000/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_CURRENT_DESKTOP=ubuntu:GNOME
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
XDG_MENU_PREFIX=gnome-
XDG_RUNTIME_DIR=/run/user/1000
XDG_SEAT=seat0
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_ID=3
XDG_SESSION_TYPE=x11
XDG_VTNR=2
XMODIFIERS=@im=ibus
ZSH=/home/san_sanchez/.oh-my-zsh
_=/home/san_sanchez/workspace/OC/sem_2/lab_04/source/task_1/./task_1
```

Рисунок 1.2 — Скришот вывода информации об окружении процесса (часть 2).

```
cmdline:
./task_1
```

Рисунок 1.3 — Скришот вывода содержимого файла cmdline.

```
fd:
0 -> /dev/pts/21)
1 -> /dev/pts/21)
2 -> /dev/pts/21)
3 -> /proc/793/fd
```

Рисунок 1.4 — Скришот вывода содержимого директории fd.

```
stat:
pid=10335
comm=(task_1)
state=R
ppid=4187
pgrp=10335
session=4187
tty_nr=34821
tpgid=10335
flags=4194304
minflt=97
cminflt=0
majflt=1
cmajflt=0
utime=0
stime=0
cutime=0
cstime=0
priority=20
nice=0
num_threads=1
itrealvalue=0
starttime=36511999
vsize=4624384
rss=192
rsslim=18446744073709551615
startcode=94465334353920
```

Рисунок 1.5 — Скришот вывода состояния процесса. (часть 1)

```
startcode=94465334353920
endcode=94465334360440
startstack=140726574953312
kstkesp=0
kstkeip=0
signal=0
blocked=0
sigignore=0
sigcatch=0
wchan=0
nswap=0
cnswap=0
exit_signal=17
processor=3
rt_priority=0
policy=0
delayacct_blkio_ticks=0
guest_time=0
cguest_time=0
start_data=94465336458552
end_data=94465336459712
start_brk=94465350205440
arg_start=140726574956492
arg_end=140726574956501
env_start=140726574956501
env_end=140726574960623
exit_code=0
```

Рисунок 1.6 — Скришот вывода состояния процесса. (часть 2)

1.2 Задание №2

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

Листинг 1.4 — Текст программы второго задания

```
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 <linux/uaccess.h>
8
9  MODULE_LICENSE("GPL");
10 MODULE_AUTHOR("Kupry");
11
12 static struct proc_dir_entry *proc_entry;
13 static struct proc_dir_entry *proc_dir;
14 static struct proc_dir_entry *proc_link;
15
16 static char *buffer;
17 static int buf_index, next_fortune;
18
19 static ssize_t fortune_write(struct file *filp, const char __user *buf,
       size_t len, loff_t *offp);
20 static ssize_t fortune_read(struct file *filp, char __user *buf, size_t
       count, loff_t *offp);
21
22 static int fortune_init(void)
23 {
24     static struct file_operations fileops =
25     {
26         .owner = THIS_MODULE,
27         .read = fortune_read,
28         .write = fortune_write
29     };
30
31     buffer = (char *)vmalloc(PAGE_SIZE);
32     if (!buffer)
33     {
34         printk(KERN_ERR "fortune: no memory for create buffer\n");
35         return -ENOMEM;
36     }
```

```
37
38      memset(buffer, 0, PAGE_SIZE);
39
40      proc_entry = proc_create("fortune", 0644, NULL, &fileops);
41      if (proc_entry == NULL)
42      {
43          vfree(buffer);
44          printk(KERN_ERR "fortune: couldn't create proc entry\n");
45          return -ENOMEM;
46      }
47
48      proc_dir = proc_mkdir("fortune_dir", NULL);
49      if (!proc_dir)
50      {
51          vfree(buffer);
52          printk(KERN_ERR "fortune: Couldn't create proc dir, symlink\n");
53          return -ENOMEM;
54      }
55
56      proc_link = proc_symlink("fortune_symlink", NULL, "/proc/fortune_dir");
57      if (!proc_link)
58      {
59          vfree(buffer);
60          printk(KERN_ERR "fortune: Couldn't create proc dir, symlink\n");
61          return -ENOMEM;
62      }
63
64      buf_index = 0;
65      next_fortune = 0;
66
67      printk(KERN_INFO "fortune: module loaded.\n");
68      return 0;
69  }
70
71  static ssize_t fortune_read(struct file *filp, char __user *buf, size_t
        count, loff_t *offp)
72  {
73      if (*offp > 0)
74      {
75          return 0;
76      }
77
78      if (next_fortune >= buf_index)
79      {
80          next_fortune = 0;
81      }
```

```
82
83      count = copy_to_user(buf, &buffer[next_fortune], count);
84      next_fortune += count;
85      *offp += count;
86
87      return count;
88  }
89
90  static ssize_t fortune_write(struct file *filp, const char __user *buf,
        size_t len, loff_t *offp)
91  {
92      int available_space = PAGE_SIZE - buf_index + 1;
93
94      if (len > (size_t)available_space)
95      {
96          printk(KERN_NOTICE "fortune: there isn't enough memory\n");
97          return -ENOSPC;
98      }
99
100     if (copy_from_user(&buffer[buf_index], buf, len))
101     {
102         printk(KERN_NOTICE "fortune: copy_to_user failed\n");
103         return -EFAULT;
104     }
105
106     buf_index += len;
107     buffer[buf_index - 1] = 0;
108
109     return len;
110 }
111
112 static void fortune_exit(void)
113 {
114     proc_remove(proc_entry);
115     proc_remove(proc_dir);
116     proc_remove(proc_link);
117
118     vfree(buffer);
119
120     printk(KERN_INFO "fortune: module unloaded.\n");
121 }
122
123 module_init(fortune_init);
124 module_exit(fortune_exit);
```

Результат:

```
san_sanchez@LEX ~/workspace/OC/sem_2/lab_04/source/task_2 make
make -C /lib/modules/4.15.0-99-generic/build M=/home/san_sanchez/workspace/OC/sem_2/lab_04/source/task_2 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-99-generic'
  CC [M]  /home/san_sanchez/workspace/OC/sem_2/lab_04/source/task_2/md.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/san_sanchez/workspace/OC/sem_2/lab_04/source/task_2/md.ko
```

Рисунок 1.7 — Скришот сборки модуля.

```
san_sanchez@LEX ~/workspace/OC/sem_2/lab_04/source/task_2 sudo insmod md.ko
[sudo] password for san_sanchez:
```

Рисунок 1.8 — Скришот загрузки модуля.

```
root@LEX:~# ls -l /proc | grep fortune
-rw-r--r--  1 root               root                              0 мая 10 05:06 fortune
dr-xr-xr-x  2 root               root                              0 мая 10 05:06 fortune_dir
lrwxrwxrwx  1 root               root                             13 мая 10 05:06 fortune_symlink -> /proc/fortune
```

Рисунок 1.9 — Скриншот созданных директории, файла и символьной ссылки.

```
root@LEX:~# echo "Hello, world!" >> /proc/fortune
root@LEX:~# cat /proc/fortune
Hello, world
```

Рисунок 1.10 — Скриншот работы файла.

```
root@LEX:~# cat /proc/fortune_symlink
Hello, world
```

Рисунок 1.11 — Скриншот работы ссылки.

ОБОСНОВАНИЕ ИСПОЛЬЗОВАНИЯ СПЕЦИАЛЬНЫХ ФУНКЦИИ

В Linux каждый процесс имеет собственное изолированное адресное пространство, то есть указатель ссылается не на уникальную позицию в физической памяти, а на позицию в адресном пространстве процесса.

Во время выполнения обычной программы адресация происходит автоматически, если же выполняется код ядра и необходимо получить доступ к странице кода ядра, то необходим буфер.

Когда мы хотим передавать информацию между процессом и кодом ядра, то соответсвующая специальная функция (copy_from_user, copy_to_user) получает указатель на буфер процесса.