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

Дисциплина Операционные системы

Тема Виртуальная файловая система /proc

Студент Набиев Ф.М.

Группа ИУ7–63Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

1 Задание №1

1.1 Условие

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

1.2 Реализация

В листинге 1.1 приведён текст программы, релизающей данное задание.

Листинг 1.1 – Задание №1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include <unistd.h>
6  #include <dirent.h>
7  #include <errno.h>
8
9  void print_environ(void);
10 void print_cmdline(void);
11 void print_stat(void);
12 void print_fd(void);
13
14 int main(void)
15 {
16     printf("environ:\n");
17     print_environ();
18     printf("\n");
19
20     printf("cmdline:\n");
21     print_cmdline();
22     printf("\n");
23
24     printf("stat:\n");
25     print_stat();
26     printf("\n");
27
28     printf("fd:\n");
29     print_fd();
```

```
30      printf("\n");
31
32      return 0;
33  }
34
35  static char* stat_fields[] = {
36      "pid", "comm", "state", "ppid", "pgrp",
37      "session", "tty_nr", "tpgid", "flags",
38      "minflt", "cminflt", "majflt", "cmajflt",
39      "utime", "stime", "cutime", "cstime",
40      "priority", "nice", "num_threads", "itrealvalue",
41      "starttime", "vsize", "rss", "rsslim",
42      "startcode", "endcode", "startstack",
43      "kstkesp", "kstkeip", "signal", "blocked",
44      "sigignore","sigcatch", "wchan", "nswap",
45      "cnswap", "exit_signal", "processor",
46      "rt_priority", "policy", "delayacct_blkio_ticks",
47      "guest_time", "cguest_time", "start_data",
48      "end_data", "start_brk", "arg_start",
49      "arg_end", "env_start", "env_end", "exit_code"
50  };
51
52  void print_file(const char *fname)
53  {
54      char buf[BUFSIZ] = { 0 };
55      FILE *fd;
56      int len, i;
57
58      if (!(fd = fopen(fname, "r")))
59      {
60          fprintf(stderr, "%s: %s\n", fname, strerror(errno));
61          exit(1);
62      }
63
64      while ((len = fread(buf, 1, sizeof(buf), fd)) > 0)
65      {
66          for (i = 0; i < len; ++i)
67              if (buf[i] == 0)
68                  buf[i] = 10;
69
70          buf[len - 0] = 0;
71          printf("%s", buf);
72      }
73
74      fclose(fd);
75  }
76
77  void print_environ(void)
```

```
78  {
79      print_file("/proc/self/environ");
80  }
81
82  void print_cmdline(void)
83  {
84      print_file("/proc/self/cmdline");
85  }
86
87  void print_stat(void)
88  {
89      char buf[BUFSIZ] = { 0 };
90      char *pch;
91      FILE *fd;
92      int i = 0;
93
94      if (!(fd = fopen("/proc/self/stat", "r")))
95      {
96          fprintf(stderr, "%s: %s\n", "/proc/self/stat", strerror(errno));
97          exit(1);
98      }
99
100     fread(buf, 1, sizeof(buf), fd);
101     pch = strtok(buf, " ");
102
103     while (pch)
104     {
105         printf("%s = %s\n", stat_fields[i++], pch);
106         pch = strtok(NULL, " ");
107     }
108
109     fclose(fd);
110 }
111
112 void print_fd(void)
113 {
114     DIR *dd;
115     struct dirent *dirp;
116     char path[BUFSIZ];
117     char link[BUFSIZ];
118
119     if (!(dd = opendir("/proc/self/fd/")))
120     {
121         fprintf(stderr, "%s: %s\n", "/proc/self/fd/", strerror(errno));
122         exit(1);
123     }
124
125     while ((dirp = readdir(dd)))
```

```
126     {
127         if (0 != strcmp(dirp->d_name, ".") &&
128             0 != strcmp(dirp->d_name, ".."))
129         {
130             snprintf(link, sizeof(link), "%s%s",
131                     "/proc/self/fd/", dirp->d_name);
132             readlink(link, path, sizeof(path));
133 
134             printf("%s -> %s\n", link, path);
135         }
136     }
137 
138     closedir(dd);
139 }
```

1.3 Демонстрация работы

На рисунках 1.1, 1.2, 1.3, 1.4 демонстрируется вывод списка окружения процесса, директории процесса, информации о процессе и содержания директории fd.

```
environ:
KDE_FULL_SESSION=true
GS_LIB=/home/faris/.fonts
TEXMFHOME=/home/faris/.texmf
PAM_KWALLET5_LOGIN=/run/user/1000/kwallet5.socket
USER=faris
LANGUAGE=en_US:en
XDG_SEAT=seat0
XDG_SESSION_TYPE=x11
SSH_AGENT_PID=2280
SHLVL=1
XCURSOR_SIZE=0
HOME=/home/faris
OLDPWD=/home/faris/Documents/Repositories/bmstu/OperatingSystems
QT4_IM_MODULE=ibus
LESS=-R
DESKTOP_SESSION=/usr/share/xsessions/plasma
ZSH=/home/faris/.oh-my-zsh
LSCOLORS=Gxfxcxdxbxegedabagacad
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
GTK_MODULES=gail:atk-bridge
KDE_SESSION_VERSION=5
PAGER=less
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
WINEPATH=C:\VIP52\BIN\WIN\32
LOGNAME=faris
GTK_IM_MODULE=ibus
_=clear
QT_AUTO_SCREEN_SCALE_FACTOR=0
WINDOWID=113246222
XDG_SESSION_CLASS=user
XTERM_SHELL=/bin/zsh
XDG_SESSION_ID=3
TERM=xterm-256color
PATH=/usr/local/texlive/2019/bin/x86_64-linux:/home/faris/.local/
```

Рис. 1.1 – environ

```
cmdline:
./a.out
```

Рис. 1.2 – cmdline

```
stat:
pid = 18231
comm = (a.out)
state = R
ppid = 18197
pgrp = 18231
session = 18093
tty_nr = 34818
tpgid = 18231
flags = 4194304
minflt = 75
cminflt = 0
majflt = 0
cmajflt = 0
utime = 0
stime = 0
cutime = 0
cstime = 0
priority = 20
nice = 0
num_threads = 1
itrealvalue = 0
starttime = 1366702
vsize = 2338816
rss = 188
rsslim = 18446744073709551615
startcode = 94761118588928
endcode = 94761118594765
startstack = 140723849290704
kstkesp = 0
kstkeip = 0
signal = 0
```

Рис. 1.3 – stat

```
fd:
/proc/self/fd/0 -> /dev/pts/21
/proc/self/fd/1 -> /dev/pts/21
/proc/self/fd/2 -> /dev/pts/21
/proc/self/fd/3 -> /proc/18231/fd093
```

Рис. 1.4 – fd

2 Задание №2

2.1 Условие

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.

В программе необходимо создать файл, поддиректорию и символическую ссылку.

2.2 Реализация

В листинге 2.1 приведён текст программы, реализующей данное задание.

Листинг 2.1 – Задание №2

```
1  #include <linux/fs.h>
2  #include <linux/kernel.h>
3  #include <linux/module.h>
4  #include <linux/string.h>
5  #include <linux/proc_fs.h>
6  #include <linux/vmalloc.h>
7  #include <linux/uaccess.h>
8
9  #define COOKIE_SIZE PAGE_SIZE
10
11 MODULE_LICENSE("GPL");
12 MODULE_AUTHOR("Faris Nabiev");
13
14 static int cookie_index;
15 static int next_fortune;
16 static char *cookie_pot;
17
18 static struct proc_dir_entry *p_entry;
19 static struct proc_dir_entry *p_dir;
20 static struct proc_dir_entry *p_link;
21
```

```
22 static ssize_t fortune_read(struct file *fd, char __user *buf,
23                              size_t len, loff_t *ppos);
24 static ssize_t fortune_write(struct file *fd, const char __user *buf,
25                               size_t len, loff_t *ppos);
26
27 static int fortune_init(void)
28 {
29     int rc;
30
31     static struct file_operations fops = {
32         .owner = THIS_MODULE,
33         .read = &fortune_read,
34         .write = &fortune_write
35     };
36
37     if (!(cookie_pot = (char *)vmalloc(COOKIE_SIZE)))
38     {
39         printk(KERN_ERR "fortune: can't alloc memory for cookie_pot!\n");
40         return -ENOMEM;
41     }
42
43     memset(cookie_pot, 0, COOKIE_SIZE);
44
45     if (!(p_entry = proc_create("fortune", 0644, NULL, &fops)))
46     {
47         printk(KERN_ERR "fortune: can't create fortune entry!\n");
48         vfree(cookie_pot);
49         return -ENOMEM;
50     }
51
52     printk(KERN_INFO "fortune: module have loaded.\n");
53
54     rc = 0;
55
56     if (!(p_dir = proc_mkdir("fortune_dir", NULL)))
57     {
58         printk(KERN_ERR "fortune: can't create fortune directory!\n");
59         rc = -ENOMEM;
60     }
61     if (!(p_link = proc_symlink("fortune_link",
62                                 NULL, "fortune")))
63     {
64         printk(KERN_ERR "fortune: can't create fortune symlink!\n");
65         rc = -ENOMEM;
66     }
67
68     return rc;
69 }
```

```
70
71  static void fortune_exit(void)
72  {
73      proc_remove(p_entry);
74      proc_remove(p_dir);
75      proc_remove(p_link);
76
77      vfree(cookie_pot);
78
79      printk(KERN_INFO "fortune: module have unloaded.\n");
80  }
81
82  static ssize_t fortune_read(struct file *fd, char __user *buf,
83                              size_t len, loff_t *ppos)
84  {
85      if (*ppos > 0)
86          return 0;
87
88      if (next_fortune >= cookie_index)
89          next_fortune = 0;
90
91      len = copy_to_user(buf, cookie_pot + next_fortune, len);
92      next_fortune += len;
93      *ppos += len;
94
95      return len;
96  }
97
98  static ssize_t fortune_write(struct file *fd, const char __user *buf,
99                               size_t len, loff_t *ppos)
100 {
101     int available_size = COOKIE_SIZE - cookie_index + 1;
102
103     if (len > (size_t)available_size)
104     {
105         printk(KERN_NOTICE "fortune: there is not enough "
106                            "memory in cookie pot!\n");
107         return -ENOSPC;
108     }
109
110     if (copy_from_user(cookie_pot + cookie_index, buf, len))
111     {
112         printk(KERN_NOTICE "fortune: copy_to_user failed!\n");
113         return -EFAULT;
114     }
115
116     cookie_index += len;
117     cookie_pot[cookie_index - 1] = 0;
```

```
118
119     return len;
120 }
121
122 module_init(fortune_init);
123 module_exit(fortune_exit);
```

2.3 Демонстрация работы

На рисунках 2.1, 2.2, 2.3 изображены сборка, загрузка модуля, проверка созданных файлов и возможности ввода-вывода при помощи команд echo, cat.

```
# make
make -C /lib/modules/4.19.0-9-amd64/build M=/home/faris/Documents/Repositories/bmstu
make[1]: Entering directory '/usr/src/linux-headers-4.19.0-9-amd64'
  CC [M]  /home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_14/task02.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_14/task02.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.0-9-amd64'
# insmod task02.ko
# lsmod | head -n 5
Module                  Size  Used by
task02                 16384  0
uinput                 20480  1
rfcomm                 86016  19
fuse                  122880  5
```

Рис. 2.1 – Сборка и загрузка модуля

```
# ls -hl /proc/ | grep fortune
-rw-r--r--  1 root             root                0 May 12 00:48 fortune
dr-xr-xr-x  2 root             root                0 May 12 00:48 fortune_dir
lrwxrwxrwx  1 root             root                7 May 12 00:48 fortune_link -> fortune
```

Рис. 2.2 – Проверка созданных файлов

```
# echo 'Hello' > /proc/fortune
# echo -e ' Wrold!\n' > /proc/fortune_link
# cat /proc/fortune
Hello Wrold!
```

Рис. 2.3 – Проверка работы

2.4 Обоснование использования специальных функций

Использование copy_to_user, copy_from_user обусловлено тем, что 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. При выполнении обычной программы адресация происходит автоматически. Если выполняется код ядра и необходимо получить доступ к странице кода ядра, то всегда нужен буфер, но когда мы хотим передавать информацию между процессом и кодом ядра, то соответсвующая функция ядра (copy_to_user, copy_from_user) получит указатель на буфер процесса.