



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	Степанов А. О.
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 `cmdline`.

Листинги

Листинг 1: Вывод содержимого файла

```
1 int print_file(const char *filename)
2 {
3     char buf[BUF_SIZE] = { 0 };
4     int len, i;
5     FILE *f;
6
7     f = fopen(filename, "r");
8
9     if (f == NULL)
10    {
11        perror("Can't open file\n");
12        return -1;
13    }
14
15    while ((len = fread(buf, 1, BUF_SIZE, f)) > 0)
16    {
17        for (i = 0; i < len; ++i)
18            if (buf[i] == 0)
19                buf[i] = 10;
20
21        buf[len - 1] = 0;
22        printf("%s", buf);
23    }
24
25    printf("\n");
26
27    fclose(f);
28    return 0;
29 }
```

Листинг 2: Вывод содержимого файла `stat`

```
1 int print_stat()
2 {
3     int i;
4     char buf[BUF_SIZE] = { 0 };
5     FILE *f = fopen("/proc/self/stat", "r");
6
7     if (f == NULL)
8     {
9         perror("Can't open stat\n");
10        return -1;
11    }
12
13    fread(buf, 1, BUF_SIZE, f);
14    char *pch = strtok(buf, "_");
15
16    i = 0;
17    while(pch != NULL)
18    {
19        printf("%s->%s\n", stat_names[i++], pch);
20        pch = strtok(NULL, "_");
21    }
22
23    fclose(f);
24    return 0;
25 }
```

Листинг 3: Вывод содержимого директории

```
1 int print_directory(const char *dirname)
2 {
3     struct dirent *dirp;
4     DIR *dp;
5     char str[BUF_SIZE] = { 0 };
6     char path[BUF_SIZE] = { 0 };
7
8     dp = opendir(dirname);
9
10    if (dp == NULL)
11    {
12        perror("Can't open dir");
13        return -1;
14    }
```

```

15
16 while ((dirp = readdir(dp)) != NULL)
17 {
18     if ((strcmp(dirp->d_name, ".") != 0) &&
19         (strcmp(dirp->d_name, "..") != 0))
20     {
21         sprintf(path, "%s%s", dirname, dirp->d_name);
22         readlink(path, str, BUF_SIZE);
23         printf("%s->%s\n", dirp->d_name, str);
24     }
25 }
26
27 closedir(dp);
28 return 0;
29 }

```

Листинг 4: Функция main

```

1 int main(int argc, char *argv[])
2 {
3     printf("ENVIRON:\n");
4     if (print_file("/proc/self/environ") < 0)
5         return -1;
6
7     printf("CMDLINE:\n");
8     if (print_file("/proc/self/cmdline") < 0)
9         return -1;
10
11     printf("STAT:\n");
12     if (print_stat() < 0)
13         return -1;
14
15     printf("FD:\n");
16     if (print_directory("/proc/self/fd/") < 0)
17         return -1;
18
19     return 0;
20 }

```

Результат работы

```
ENVIRON:
LANG=en_US.UTF-8
DISPLAY=:1
XDG_VTNR=2
LOGNAME=parallels
PWD=/home/parallels/Desktop/Parallels Shared Folders/Home/Repositories/operating-systems/sem_02/lab_04/part_01
XAUTHORITY=/run/user/1000/gdm/Xauthority
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
QT_QPA_PLATFORMTHEME=qt5qt6
JOURNAL_STREAM=8:20745
COLORTERM=truecolor
XDG_SESSION_ID=6
DESKTOP_SESSION=gnome
XDG_SESSION_DESKTOP=gnome
GDMSESSION=gnome
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
USERNAME=parallels
WINDOWPATH=2
DBUS_SESSION_BUS_ADDRESS=unix:ath=/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=29360133
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
SSH_AGENT_PID=1440
HOME=/home/parallels
```

Рис. 1: Вывод информации об окружении процесса (часть 1)

```
XDG_SEAT=seat0
XDG_UNTIME_DIR=/run/user/1000
SESSION_MANAGER=local/debian-gnu-linux-vm:/tmp/.ICE-unix/1394,unix/debian-gnu-linux-vm:/tmp/.ICE-unix/1394
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
USER=parallels
SHLVL=1
OLDPWD=/home/parallels/Desktop/Parallels Shared Folders/Home/Repositories/operating-systems/sem_02/lab_04
ZSH=/home/parallels/.oh-my-zsh
PAGER=less
LESS=-R
LSCOLORS=Gxfxcxdxbxegedabagacad
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd=40;33:01:or=40;31:01:mi=00:su=37;1:sg=30;43:ca=30;41:tw=30;
42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lzh=01;31:*.lzm=01;31:*.
tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz
=01;31:*.zst=01;31:*.tzt=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.
ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;3
5:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.t
iff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=0
1;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.f
lc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.
ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;3
6:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
=/home/parallels/Desktop/Parallels Shared Folders/Home/Repositories/operating-systems/sem_02/lab_04/part_01/./a.out
```

Рис. 2: Вывод информации об окружении процесса (часть 2)

```
CMDLINE:  
./a.out
```

Рис. 3: Вывод директории процесса (файл `cmdline`)

```
FD:  
0 -> /dev/pts/0  
1 -> /dev/pts/0  
2 -> /dev/pts/0  
3 -> /proc/22274/fd
```

Рис. 4: Вывод содержания директории `fd`

```
STAT:
pid -> 4574
comm -> (a.out)
state -> R
ppid -> 1997
pgrp -> 4574
session -> 1997
tty_nr -> 34816
tpgid -> 4574
flags -> 4194304
minflt -> 90
cminflt -> 0
majflt -> 3
cmajflt -> 0
utime -> 0
stime -> 0
cutime -> 0
cstime -> 0
priority -> 20
nice -> 0
num_threads -> 1
itrealvalue -> 0
starttime -> 84822
vsize -> 4263936
rss -> 168
rsslim -> 18446744073709551615
startcode -> 94057308016640
endcode -> 94057308022772
startstack -> 140733068092000
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
delayacct_blkio_ticks -> 0
guest_time -> 0
cguest_time -> 0
start_data -> 94057310121432
end_data -> 94057310122592
start_brk -> 94057327190016
arg_start -> 140733068100650
arg_end -> 140733068100658
env_start -> 140733068100658
env_end -> 140733068103664
exit_code -> 0
```

Рис. 5: Вывод состояния процесса (файл stat)

ЧАСТЬ 2

Задание

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.

Листинги

Листинг 5: Код модуля

```
1 #include <linux/module.h>
2 #include <linux/moduleparam.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5 #include <linux/proc_fs.h>
6 #include <asm/uaccess.h>
7 #include <linux/vmalloc.h>
8
9 #define MAX_COOKIE_LENGTH PAGE_SIZE
10
11 MODULE_LICENSE("Dual_BSD/GPL");
12 MODULE_AUTHOR("Alexander_Stepanov");
13
14 static struct proc_dir_entry *proc_entry;
15 static struct proc_dir_entry *proc_directory;
16 static struct proc_dir_entry *proc_link;
17 static char *cookie_pot;
18 static int cookie_index;
19 static int next_fortune;
20
21 static ssize_t fortune_write(struct file *f, const char __user *buff,
22     size_t len, loff_t *data)
23 {
24     int space_available = (MAX_COOKIE_LENGTH - cookie_index) + 1;
25
26     if (len > space_available)
27     {
28         printk(KERN_INFO "fortune: cookie_pot is full!\n");
```



```

29         return -ENOSPC;
30     }
31
32     if (copy_from_user(&cookie_pot[cookie_index], buff, len))
33     {
34         return -EFAULT;
35     }
36
37     cookie_index += len;
38     cookie_pot[cookie_index-1] = 0;
39
40     return len;
41 }
42
43 static ssize_t fortune_read(struct file *f, char __user *buff,
44     size_t len, loff_t *data)
45 {
46     if (*data > 0) return 0;
47
48     if (next_fortune >= cookie_index)
49         next_fortune = 0;
50
51     len = copy_to_user(buff, &cookie_pot[next_fortune], len);
52     next_fortune += len;
53
54     *data = 1;
55
56     return len;
57 }
58
59 static struct file_operations ops =
60 {
61     .owner = THIS_MODULE,
62     .read = fortune_read,
63     .write = fortune_write,
64 };
65
66 static int md_init(void)
67 {
68     int ret = 0;
69     cookie_pot = (char *)vmalloc( MAX_COOKIE_LENGTH);
70

```

```

71     if (!cookie_pot)
72     {
73         ret = -ENOMEM;
74     }
75     else
76     {
77         memset(cookie_pot, 0, MAX_COOKIE_LENGTH );
78         proc_entry = proc_create("fortune", 0644, NULL, &ops);
79
80         if (proc_entry == NULL)
81         {
82             ret = -ENOMEM;
83             vfree(cookie_pot);
84             printk(KERN_INFO "fortune:␣Couldn't␣create␣proc␣entry\n");
85         }
86         else
87         {
88             cookie_index = 0;
89             next_fortune = 0;
90             printk(KERN_INFO "fortune:␣Module␣loaded.\n");
91
92             proc_directory = proc_mkdir("fortune_dir", NULL);
93
94             if (proc_directory == NULL)
95             {
96                 ret = -ENOMEM;
97                 printk(KERN_ERR "fortune:␣Couldn't␣create␣dir");
98             }
99
100            proc_link = proc_symlink("fortune_link", NULL, "fortune");
101
102            if (proc_link == NULL)
103            {
104                ret = -ENOMEM;
105                printk(KERN_ERR "fortune:␣Couldn't␣create␣symlink");
106            }
107        }
108    }
109
110    return ret;
111 }
112

```

```

113 static void md_exit(void)
114 {
115     proc_remove(proc_entry);
116     proc_remove(proc_directory);
117     proc_remove(proc_link);
118     vfree(cookie_pot);
119     printk(KERN_INFO "fortune: Module unloaded.\n");
120 }
121
122 module_init(md_init);
123 module_exit(md_exit);

```

Результат работы

```

debian-gnu-linux-vm# make
make -C /lib/modules/4.9.0-12-amd64/build M=/media/psf/Home/Repositories/operating-systems/sem_02/lab_04/part_02 modules
make[1]: Entering directory '/usr/src/linux-headers-4.9.0-12-amd64'
  CC [M]  /media/psf/Home/Repositories/operating-systems/sem_02/lab_04/part_02/md.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /media/psf/Home/Repositories/operating-systems/sem_02/lab_04/part_02/md.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.0-12-amd64'
debian-gnu-linux-vm# insmod md.ko
debian-gnu-linux-vm#

```

Рис. 6: Сборка и загрузка модуля

```

debian-gnu-linux-vm# lsmod
Module                  Size  Used by
md                      16384  0
fuse                   98304  3

```

Рис. 7: Модуль загружен

```

debian-gnu-linux-vm# ls -l /proc | grep fortune
-rw-r--r--  1 root      root      0 Mar 21 21:28 fortune
dr-xr-xr-x  2 root      root      0 Mar 21 21:28 fortune_dir
lrwxrwxrwx  1 root      root      7 Mar 21 21:28 fortune_link -> fortune
debian-gnu-linux-vm#

```

Рис. 8: Модуль создал файл, символическую ссылку и директорию

```

debian-gnu-linux-vm# echo "test1\n" > /proc/fortune
debian-gnu-linux-vm# echo "test2\n" > /proc/fortune
debian-gnu-linux-vm# cat /proc/fortune
test1
test2

```

Рис. 9: Проверка работы файла

```
test2
debian-gnu-linux-vm# cat /proc/fortune_link
test1
test2
debian-gnu-linux-vm#
```

Рис. 10: Проверка работы ссылки

Обоснование использования специальных функций

В linux память сегментирована, это значит, что указатель не ссылается на уникальную позицию в памяти, а ссылается на позицию в сегменте. Процессу доступен только собственный сегмент памяти. Если выполняется обычная программа, то адресация происходит автоматически. Если выполняется код ядра и необходимо получить доступ к сегменту кода ядра, то всегда нужен буфер, но когда мы хотим передать информацию между процессом и кодом ядра, то соответствующая функция ядра получит указатель на буфер процесса.