



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

По предмету: «Операционные системы»

Тема: Файловая система /proc

Преподаватель: Рязанова Н.Ю.

Студент: Мирзоян С.А.

Группа: ИУ7-65Б

Москва, 2020 г.

Часть I

Содержимое файла /proc/[pid]/environ

Окружение (environment) или среда — это набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, доступный каждому пользовательскому процессу. Иными словами, окружение — это набор переменных окружения.

Некоторые переменные окружения:

- LS_COLORS - используется для определения цветов, с которыми будут выведены имена файлов при вызове ls.
- LESSCLOSE, LESSOPEN – определяют пре- и пост- обработчики файла, который открывается при вызове less.
- XDG_MENU_PREFIX, XDG_VTNR, XDG_SESSION_ID, XDG_SESSION_TYPE, XDG_DATA_DIRS, XDG_SESSION_DESKTOP, XDG_CURRENT_DESKTOP, XDG_RUNTIME_DIR, XDG_CONFIG_DIRS, DESKTOP_SESSION – переменные, необходимые для вызова xdg-open, использующейся для открытия файла или URL в пользовательском приложении.
- LANG – язык и кодировка пользователя.
- DISPLAY – указывает приложениям, куда отобразить графический пользовательский интерфейс.
- GNOME_SHELL_SESSION_MODE, GNOME_TERMINAL_SCREEN, GNOME_DESKTOP_SESSION_ID, GNOME_TERMINAL_SERVICE, GJS_DEBUG_OUTPUT, GJS_DEBUG_TOPICS, GTK_MODULES, GTK_IM_MODULE, VTE_VERSION – переменные среды рабочего стола GNOME.
- COLORTERM – определяет поддержку 24-битного цвета.
- USER – имя пользователя, от чьего имени запущен процесс,
- USERNAME – имя пользователя, кто инициировал запуск процесса.
- SSH_AUTH_SOCK - путь к сокету, который агент использует для коммуникации с другими процессами.
- TEXTDOMAINDIR, TEXTDOMAIN – директория и имя объекта сообщения, получаемого при вызове gettext.
- PWD – путь к рабочей директории.
- HOME – путь к домашнему каталогу текущего пользователя.
- SSH_AGENT_PID - идентификатор процесса ssh-agent.
- TERM – тип запущенного терминала.
- SHELL – путь к предпочтительной оболочке командной строки.
- SHLVL – уровень текущей командной оболочки.
- LOGNAME – имя текущего пользователя.
- PATH - список каталогов, в которых система ищет исполняемые файлы.

- `_` - полная командная строка процесса
- `OLDPWD` - путь к предыдущему рабочему каталогу.

1. Листинг программы для вывода информации об окружении процесса:

```
1. #include <stdio.h>
2. #define BUFSIZE 0x1000
3.
4. int main(int argc, char* argv[])
5. {
6.     char buffer[BUFSIZE];
7.     int len;
8.     int i;
9.     FILE* f;
10.
11.     f = fopen("/proc/self/environ", "r");
12.     while ((len = fread(buffer, 1, BUFSIZE, f)) > 0)
13.     {
14.         for (i = 0; i < len; i++)
15.             if (buffer[i] == 0)
16.                 buffer[i] = 10;
17.         buffer[len - 1] = 10;
18.
19.         if (i % 1000 == 0)
20.             printf("0x%08x\n", i);
21.         if (i % 1000 == 0)
22.             printf("0x%08x\n", len);
23.         if (i % 1000 == 0)
24.             printf("0x%08x\n", len - i);
25.         if (i % 1000 == 0)
26.             printf("0x%08x\n", len - i - 1);
27.         if (i % 1000 == 0)
28.             printf("0x%08x\n", len - i - 2);
29.         if (i % 1000 == 0)
30.             printf("0x%08x\n", len - i - 3);
31.         if (i % 1000 == 0)
32.             printf("0x%08x\n", len - i - 4);
33.         if (i % 1000 == 0)
34.             printf("0x%08x\n", len - i - 5);
35.         if (i % 1000 == 0)
36.             printf("0x%08x\n", len - i - 6);
37.         if (i % 1000 == 0)
38.             printf("0x%08x\n", len - i - 7);
39.         if (i % 1000 == 0)
40.             printf("0x%08x\n", len - i - 8);
41.         if (i % 1000 == 0)
42.             printf("0x%08x\n", len - i - 9);
43.         if (i % 1000 == 0)
44.             printf("0x%08x\n", len - i - 10);
45.         if (i % 1000 == 0)
46.             printf("0x%08x\n", len - i - 11);
47.         if (i % 1000 == 0)
48.             printf("0x%08x\n", len - i - 12);
49.         if (i % 1000 == 0)
50.             printf("0x%08x\n", len - i - 13);
51.         if (i % 1000 == 0)
52.             printf("0x%08x\n", len - i - 14);
53.         if (i % 1000 == 0)
54.             printf("0x%08x\n", len - i - 15);
55.         if (i % 1000 == 0)
56.             printf("0x%08x\n", len - i - 16);
57.         if (i % 1000 == 0)
58.             printf("0x%08x\n", len - i - 17);
59.         if (i % 1000 == 0)
60.             printf("0x%08x\n", len - i - 18);
61.         if (i % 1000 == 0)
62.             printf("0x%08x\n", len - i - 19);
63.         if (i % 1000 == 0)
64.             printf("0x%08x\n", len - i - 20);
65.         if (i % 1000 == 0)
66.             printf("0x%08x\n", len - i - 21);
67.         if (i % 1000 == 0)
68.             printf("0x%08x\n", len - i - 22);
69.         if (i % 1000 == 0)
70.             printf("0x%08x\n", len - i - 23);
71.         if (i % 1000 == 0)
72.             printf("0x%08x\n", len - i - 24);
73.         if (i % 1000 == 0)
74.             printf("0x%08x\n", len - i - 25);
75.         if (i % 1000 == 0)
76.             printf("0x%08x\n", len - i - 26);
77.         if (i % 1000 == 0)
78.             printf("0x%08x\n", len - i - 27);
79.         if (i % 1000 == 0)
80.             printf("0x%08x\n", len - i - 28);
81.         if (i % 1000 == 0)
82.             printf("0x%08x\n", len - i - 29);
83.         if (i % 1000 == 0)
84.             printf("0x%08x\n", len - i - 30);
85.         if (i % 1000 == 0)
86.             printf("0x%08x\n", len - i - 31);
87.         if (i % 1000 == 0)
88.             printf("0x%08x\n", len - i - 32);
89.         if (i % 1000 == 0)
90.             printf("0x%08x\n", len - i - 33);
91.         if (i % 1000 == 0)
92.             printf("0x%08x\n", len - i - 34);
93.         if (i % 1000 == 0)
94.             printf("0x%08x\n", len - i - 35);
95.         if (i % 1000 == 0)
96.             printf("0x%08x\n", len - i - 36);
97.         if (i % 1000 == 0)
98.             printf("0x%08x\n", len - i - 37);
99.         if (i % 1000 == 0)
100.             printf("0x%08x\n", len - i - 38);
101.         if (i % 1000 == 0)
102.             printf("0x%08x\n", len - i - 39);
103.         if (i % 1000 == 0)
104.             printf("0x%08x\n", len - i - 40);
105.         if (i % 1000 == 0)
106.             printf("0x%08x\n", len - i - 41);
107.         if (i % 1000 == 0)
108.             printf("0x%08x\n", len - i - 42);
109.         if (i % 1000 == 0)
110.             printf("0x%08x\n", len - i - 43);
111.         if (i % 1000 == 0)
112.             printf("0x%08x\n", len - i - 44);
113.         if (i % 1000 == 0)
114.             printf("0x%08x\n", len - i - 45);
115.         if (i % 1000 == 0)
116.             printf("0x%08x\n", len - i - 46);
117.         if (i % 1000 == 0)
118.             printf("0x%08x\n", len - i - 47);
119.         if (i % 1000 == 0)
120.             printf("0x%08x\n", len - i - 48);
121.         if (i % 1000 == 0)
122.             printf("0x%08x\n", len - i - 49);
123.         if (i % 1000 == 0)
124.             printf("0x%08x\n", len - i - 50);
125.         if (i % 1000 == 0)
126.             printf("0x%08x\n", len - i - 51);
127.         if (i % 1000 == 0)
128.             printf("0x%08x\n", len - i - 52);
129.         if (i % 1000 == 0)
130.             printf("0x%08x\n", len - i - 53);
131.         if (i % 1000 == 0)
132.             printf("0x%08x\n", len - i - 54);
133.         if (i % 1000 == 0)
134.             printf("0x%08x\n", len - i - 55);
135.         if (i % 1000 == 0)
136.             printf("0x%08x\n", len - i - 56);
137.         if (i % 1000 == 0)
138.             printf("0x%08x\n", len - i - 57);
139.         if (i % 1000 == 0)
140.             printf("0x%08x\n", len - i - 58);
141.         if (i % 1000 == 0)
142.             printf("0x%08x\n", len - i - 59);
143.         if (i % 1000 == 0)
144.             printf("0x%08x\n", len - i - 60);
145.         if (i % 1000 == 0)
146.             printf("0x%08x\n", len - i - 61);
147.         if (i % 1000 == 0)
148.             printf("0x%08x\n", len - i - 62);
149.         if (i % 1000 == 0)
150.             printf("0x%08x\n", len - i - 63);
151.         if (i % 1000 == 0)
152.             printf("0x%08x\n", len - i - 64);
153.         if (i % 1000 == 0)
154.             printf("0x%08x\n", len - i - 65);
155.         if (i % 1000 == 0)
156.             printf("0x%08x\n", len - i - 66);
157.         if (i % 1000 == 0)
158.             printf("0x%08x\n", len - i - 67);
159.         if (i % 1000 == 0)
160.             printf("0x%08x\n", len - i - 68);
161.         if (i % 1000 == 0)
162.             printf("0x%08x\n", len - i - 69);
163.         if (i % 1000 == 0)
164.             printf("0x%08x\n", len - i - 70);
165.         if (i % 1000 == 0)
166.             printf("0x%08x\n", len - i - 71);
167.         if (i % 1000 == 0)
168.             printf("0x%08x\n", len - i - 72);
169.         if (i % 1000 == 0)
170.             printf("0x%08x\n", len - i - 73);
171.         if (i % 1000 == 0)
172.             printf("0x%08x\n", len - i - 74);
173.         if (i % 1000 == 0)
174.             printf("0x%08x\n", len - i - 75);
175.         if (i % 1000 == 0)
176.             printf("0x%08x\n", len - i - 76);
177.         if (i % 1000 == 0)
178.             printf("0x%08x\n", len - i - 77);
179.         if (i % 1000 == 0)
180.             printf("0x%08x\n", len - i - 78);
181.         if (i % 1000 == 0)
182.             printf("0x%08x\n", len - i - 79);
183.         if (i % 1000 == 0)
184.             printf("0x%08x\n", len - i - 80);
185.         if (i % 1000 == 0)
186.             printf("0x%08x\n", len - i - 81);
187.         if (i % 1000 == 0)
188.             printf("0x%08x\n", len - i - 82);
189.         if (i % 1000 == 0)
190.             printf("0x%08x\n", len - i - 83);
191.         if (i % 1000 == 0)
192.             printf("0x%08x\n", len - i - 84);
193.         if (i % 1000 == 0)
194.             printf("0x%08x\n", len - i - 85);
195.         if (i % 1000 == 0)
196.             printf("0x%08x\n", len - i - 86);
197.         if (i % 1000 == 0)
198.             printf("0x%08x\n", len - i - 87);
199.         if (i % 1000 == 0)
200.             printf("0x%08x\n", len - i - 88);
201.         if (i % 1000 == 0)
202.             printf("0x%08x\n", len - i - 89);
203.         if (i % 1000 == 0)
204.             printf("0x%08x\n", len - i - 90);
205.
```

```
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
QT_IM_MODULE=ibus
QT4_IM_MODULE=tbus
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@ln=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=2913
GTK_MODULES=gall:atk-bridge
PWD=/home/sergey/Рабочий стол
LOGNAME=sergey
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITYTYV=/run/user/1000/gdm/Xauthority
GJS_DEBUG_TOPICS=:JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/sergey
USERNAME=sergey
TM_CONFIG_PATH=/e
LANG=en_RU.UTF-8
LS_COLORS=rs=0:di=01:34:ln=01:36:nh=00:pl=40:33:so=01:35:do=01:35:bd=40:33:01:cd=40:33:01:or=40:31:01:ml=00:su=37:41:sg=30:43:ca=30:41:tw=30:42:ow=34:42:st=37:44:ex=01:32:*.*tar=01:31*:.*tgr=01:31*:.*arc=01:31*:.*x=01:31*:.*z=01:31*:.*taz=01:31*:.*lha=01:31*:.*lz4=01:31*:.*lzh=01:31*:.*lzna=01:31*:.*tlz=01:31*:.*txz=01:31*:.*tzo=01:31*:.*t7z=01:31*:.*zip=01:31*:.*z=01:31*:.*d=01:31*:.*gz=01:31*:.*lrz=01:31*:.*lz=01:31*:.*lzo=01:31*:.*xz=01:31*:.*zt=01:31*:.*tzt=01:31*:.*bz2=01:31*:.*bz=01:31*:.*tbz=01:31*:.*tbz2=01:31*:.*t=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*:.*xpi=01:31*:.*cpio=01:31*:.*7z=01:31*:.*rz=01:31*:.*cab=01:31*:.*wim=01:31*:.*swm=01:31*:.*odm=01:31*:.*odp=01:31*:.*jpegu=01:35*:.*njpg=01:35*:.*gif=01:35*:.*bmp=01:35*:.*pbm=01:35*:.*pgm=01:35*:.*ppm=01:35*:.*qpm=01:35*:.*xbm=01:35*:.*xpm=01:35*:.*tif=01:35*:.*tiff=01:35*:.*png=01:35*:.*svgz=01:35*:.*mng=01:35*:.*pcx=01:35*:.*mpg=01:35*:.*mpeg=01:35*:.*m2v=01:35*:.*mkv=01:35*:.*webm=01:35*:.*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*:.*flc=01:35*:.*avi=01:35*:.*fl=01:35*:.*flv=01:35*:.*gl=01:35*:.*dl=01:35*:.*xcf=01:35*:.*xwd=01:35*:.*yuv=01:35*:.*cgm=01:35*:.*enf=01:35*:.*ogv=01:35*:.*ogx=01:35*:.*aac=00:36*:.*au=00:36*:.*flac=00:36*:.*m4a=00:36*:.*nld=00:36*:.*nldl=00:36*:.*nka=00:36*:.*mp3=00:36*:.*npc=00:36*:.*ogg=00:36*:.*ra=00:36*:.*wav=00:36*:.*oga=00:36*:.*opus=00:36*:.*spx=00:36*:.*xspf=00:36*
XDG_CURRENT_DESKTOP=ubuntu:GNOME
YTE_VERSION=5802
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/9d5be677_97ab_4ac3_B93d_55b40b50abff
INVOCATION_ID=3605ba2c589947fe8bb883c4c0a05a65
MANAGERPID=2753
CLUTTER_IM_MODULE=ibus
GJS_DEBUG_OUTPUT=stderr
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=[ /usr/bin/lesspipe %s
USER=sergey
GNOME_TERMINAL_SERVICE=:1.170
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:29182
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/napd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
_=./lab4
```

Рис.1 Вывод информации об окружении процесса

Содержимое файла /proc/[pid]/stat:

```
sergey@sergey-VirtualBox:~/lab4$ ./lab4_stat
stat:
1) 17545
2) (lab4_stat)
3) R
4) 14349
5) 17545
6) 14349
7) 34817
8) 17545
9) 4194304
10) 73
11) 0
12) 0
13) 0
14) 0
15) 0
16) 0
17) 0
18) 20
19) 0
20) 1
21) 0
22) 319396
23) 2539520
24) 176
25) 18446744073709551615
26) 94746994098176
27) 94746994103221
28) 140724904381312
29) 0
30) 0
31) 0
32) 0
33) 0
34) 0
35) 0
36) 0
37) 0
38) 17
39) 2
40) 0
41) 0
42) 0
43) 0
44) 0
45) 94746994113928
46) 94746994114576
47) 94747014680576
48) 140724904387480
49) 140724904387492
50) 140724904387492
51) 140724904390636
52) 0
```

Рис.2 Вывод информации, характеризующей состояние процесса

- 1) pid - уникальный идентификатор процесса.
- 2) comm - имя исполняемого файла в круглых скобках.
- 3) state - состояние процесса.
- 4) ppid - уникальный идентификатор процесса-предка.
- 5) pgrp - уникальный идентификатор группы.
- 6) session - уникальный идентификатор сессии.

- 7) `tty_nr` – управляющий терминал.
- 8) `tpgid` – уникальный идентификатор группы управляющего терминала.
- 9) `flags` – флаги.
- 10) `minflt` - Количество незначительных сбоев, которые возникли при выполнении процесса, и которые не требуют загрузки страницы памяти с диска.
- 11) `cminflt` - количество незначительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.
- 12) `majflt` - количество значительных сбоев, которые возникли при работе процесса, и которые потребовали загрузки страницы памяти с диска.
- 13) `cmajflt` - количество значительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.
- 14) `utime` - количество тиков, которые данный процесс провел в режиме пользователя.
- 15) `stime` - количество тиков, которые данный процесс провел в режиме ядра.
- 16) `cutime` - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме пользователя.
- 17) `cstime` - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме ядра.
- 18) `priority` – для процессов реального времени это отрицательный приоритет планирования минус один, то есть число в диапазоне от -2 до -100, соответствующее приоритетам в реальном времени от 1 до 99. Для остальных процессов это необработанное значение `nice`, представленное в ядре. Ядро хранит значения `nice` в виде чисел в диапазоне от 0 (высокий) до 39 (низкий), соответствующих видимому пользователю диапазону от -20 до 19.
- 19) `nice` - значение для `nice` в диапазоне от 19 (наиболее низкий приоритет) до -20 (наивысший приоритет).
- 20) `num_threads` – число потоков в данном процессе.
- 21) `itrealvalue` – количество мигнов до того, как следующий `SIGALARM` будет послан процессу интервальным таймером. С ядра версии 2.6.17 больше не поддерживается и установлено в 0.
- 22) `starttime` - время в тиках запуска процесса после начальной загрузки системы.
- 23) `vsizе` - размер виртуальной памяти в байтах.
- 24) `rss` - резидентный размер: количество страниц, которые занимает процесс в памяти. Это те страницы, которые заняты кодом, данными и пространством стека. Сюда не включаются страницы, которые не были загружены по требованию или которые находятся в своппинге.

- 25) rsslim - текущий лимит в байтах на резидентный размер процесса.
- 26) startcode - адрес, выше которого может выполняться код программы.
- 27) endcode - адрес, ниже которого может выполняться код программ.
- 28) startstack - адрес начала стека.
- 29) kstkesp - текущее значение ESP (указателя стека).
- 30) kstkeip - текущее значение EIP (указатель команд).
- 31) signal - битовая карта ожидающих сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
- 32) blocked - битовая карта блокируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
- 33) sigignore - битовая карта игнорируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
- 34) sigcatch - битовая карта перехватываемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
- 35) wchan - "канал", в котором ожидает процесс.
- 36) nswap - количество страниц на своппинге (не обслуживается).
- 37) cnsvar - суммарное nswap для процессов-потомков (не обслуживается).
- 38) exit_signal - сигнал, который будет послан предку, когда процесс завершится.
- 39) processor - номер процессора, на котором последний раз выполнялся процесс.
- 40) rt_priority - приоритет планирования реального времени, число в диапазоне от 1 до 99 для процессов реального времени, 0 для остальных.
- 41) policy - политика планирования.
- 42) delayacct_blkio_ticks - суммарные задержки ввода/вывода в тиках.
- 43) guest_time – гостевое время процесса (время, потраченное на выполнение виртуального процессора на гостевой операционной системе) в тиках.
- 44) cguest_time - гостевое время для потомков процесса в тиках.
- 45) start_data - адрес, выше которого размещаются инициализированные и неинициализированные (BSS) данные программы.

- 46) `end_data` - адрес, ниже которого размещаются инициализированные и неинициализированные (BSS) данные программы.
- 47) `start_brk` - адрес, выше которого куча программы может быть расширена с использованием `brk()`.
- 48) `arg_start` - адрес, выше которого размещаются аргументы командной строки (`argv`).
- 49) `arg_end` - адрес, ниже которого размещаются аргументы командной строки (`argv`).
- 50) `env_start` - адрес, выше которого размещается окружение программы.
- 51) `env_end` - адрес, ниже которого размещается окружение программы.
- 52) `exit_code` – статус завершения потока в форме, возвращаемой `waitpid()`.

2. Листинг программы для вывода информации, характеризующей состояние процесса:

```
1. #include <stdio.h>
2. #include <string.h>
3.
4. #define BUFFSIZE 0x1000
5.
6. int main(int argc, char *argv)
7. {
8.     char buffer[BUFFSIZE];
9.     int len;
10.    FILE *f;
11.
12.    f = fopen("/proc/self/stat", "r");
13.    fread(buffer, 1, BUFFSIZE, f);
14.    char* p_ch = strtok(buffer, " ");
15.
16.    printf("stat: \n");
17.
18.    while (p_ch != NULL)
19.    {
20.        printf("%s \n", p_ch);
21.        p_ch = strtok(NULL, " ");
22.    }
23.    fclose(f);
24.    return 0;
25. }
```

Содержимое файла fd

3. Листинг программы для вывода содержимого директории fd.

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <dirent.h>
4. #include <unistd.h>
5.
6. #define BUFFSIZE 0x1000
7.
8. int main(int argc, char* argv)
9. {
10.     struct dirent *dirp;
11.     DIR *dp;
12.
13.     char string[BUFFSIZE];
14.     char path[BUFFSIZE];
15.
16.     dp = opendir("/proc/self/fd");
17.     puts("\t FD \n ");
18.
19.     while ((dirp = readdir(dp)) != NULL)
20.     {
21.         if ((strcmp(dirp->d_name, ".") != 0 ) && (strcmp(dirp->d_name,
22.             "..") != 0))
23.         {
24.             sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
25.             readlink(path, string, BUFFSIZE);
26.             path[BUFFSIZE] = '\0';
27.             printf("%s -> %s\n", dirp->d_name, string);
28.         }
29.     }
30.     closedir(dp);
31.     return 0;
31. }
```



```
FD
0 -> /dev/pts/0
1 -> /dev/pts/0
2 -> /dev/pts/0
3 -> /proc/9447/fd
```

Рис. 2 Результат работы программы, содержимое директории fd

Содержимое директории cmdline

4. Листинг программы для вывода содержимого директории cmdline.

```
1. #include <stdio.h>
2. #include <unistd.h>
3.
4. #define BUFFSIZE 0x1000
5.
6. int main(int argc, char *argv)
7. {
8.     char buffer[BUFFSIZE];
9.     FILE *f;
10.    int len;
11.
12.    f = fopen("/proc/self/cmdline", "r");
13.    len = fread(buffer, 1, BUFFSIZE, f);
14.    buffer[--len] = 0;
15.
16.    printf("cmdline for %d \nprocess = %s\n", getpid(), buffer);
17.    fclose(f);
18.
19.    return 0;
20. }
```

```
cmdline for 9896
process = ./lab4_cmdline
```

Рис. 3 Результат работы программы, содержимое директории cmdline

Часть II

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.

Листинг программы:

```
1. #include <linux/module.h>
2. #include <linux/init.h>
3. #include <linux/kernel.h>
4. #include <linux/proc_fs.h>
5. #include <linux/string.h>
6. #include <linux/vmalloc.h>
7. #include <asm/uaccess.h>
8. #include <linux/uaccess.h>
9.
10. #define COOKIE_BUF_SIZE PAGE_SIZE
11.
12.
13. MODULE_LICENSE("GPL");
14. MODULE_AUTHOR("Sergey Mirzoyan");
15.
16. ssize_t fortune_read(struct file *file, char *buf, size_t count, loff_t
    *f_pos);
17. ssize_t fortune_write(struct file *file, const char *buf, size_t count,
    loff_t *f_pos);
18. int fortune_init(void);
19. void fortune_exit(void);
20.
21. struct file_operations fops = {
22.     .owner = THIS_MODULE,
23.     .read = fortune_read,
24.     .write = fortune_write,
25. };
26.
27.
```

```
28. char *cookie_buf;
29. struct proc_dir_entry *proc_file;
30. unsigned int read_index;
31. unsigned int write_index;
32.
33. ssize_t fortune_read(struct file *file, char *buf, size_t count, loff_t
    *f_pos)
34. {
35.     int len;
36.
37.     //there's no fortune or a fortune has already been read
38.     //the *f_pos > 0 hack is needed because `cat /proc/fortune` would
    otherwise
39.     //display every thing in the cookie_buf
40.
41.     if (write_index == 0 || *f_pos > 0)
42.     {
43.         return 0;
44.     }
45.
46.     // cicle through fortunes
47.
48.     if (read_index >= write_index)
49.     {
50.         read_index = 0;
51.     }
52.
53.     len = sprintf(buf, "%s\n", &cookie_buf[read_index]);
54.     read_index += len;
55.     *f_pos += len;
56.
57.     return len;
58. }
59.
60. ssize_t fortune_write(struct file *file, const char *buf, size_t count,
    loff_t *f_pos)
61. {
62.     int free_space = (COOKIE_BUF_SIZE - write_index) + 1;
63.
64.     if (count > free_space)
```

```
65.     {
66.         printk(KERN_INFO "Cookie pot full.\n");
67.         return -ENOSPC;
68.     }
69.
70.     if (copy_from_user(&cookie_buf[write_index], buf, count))
71.     {
72.         return -EFAULT;
73.     }
74.
75.     write_index += count;
76.     cookie_buf[write_index-1] = 0;
77.
78.     return count;
79. }
80.
81.
82. int fortune_init(void)
83. {
84.     cookie_buf = vmalloc(COOKIE_BUF_SIZE);
85.
86.     if (!cookie_buf)
87.     {
88.         printk(KERN_INFO "Not enough memory for the cookie pot.\n");
89.         return -ENOMEM;
90.     }
91.
92.     memset(cookie_buf, 0, COOKIE_BUF_SIZE);
93.     proc_file = proc_create("fortune", 0666, NULL, &fops);
94.
95.     if (!proc_file)
96.     {
97.         vfree(cookie_buf);
98.         printk(KERN_INFO "Cannot create fortune file.\n");
99.         return -ENOMEM;
100.    }
101.
102.    read_index = 0;
103.    write_index = 0;
104.
```

```

105.     proc_mkdir("Dir_in_proc", NULL);
106.     proc_symlink("Symbolic_in_proc", NULL, "/proc/fortune");
107.
108.     printk(KERN_INFO "Fortune module loaded.\n");
109.     return 0;
110. }
111.
112.
113. void fortune_exit(void)
114. {
115.     remove_proc_entry("fortune", NULL);
116.
117.     if (cookie_buf)
118.     {
119.         vfree(cookie_buf);
120.     }
121.
122.     printk(KERN_INFO "Fortune module unloaded.\n");
123. }
124.
125. module_init(fortune_init);
126. module_exit(fortune_exit);

```

Результат работы программы.

```

sergey@sergey-VirtualBox:~/lab4/fort$ sudo insmod fortune.ko
sergey@sergey-VirtualBox:~/lab4/fort$ dmesg | tail -1
[ 4079.351503] Fortune module loaded.
sergey@sergey-VirtualBox:~/lab4/fort$ echo "Hi, my name is Sergey" > /proc/fortune
sergey@sergey-VirtualBox:~/lab4/fort$ cat /proc/fortune
Hi, my name is Sergey
sergey@sergey-VirtualBox:~/lab4/fort$ sudo rmmod fortune
sergey@sergey-VirtualBox:~/lab4/fort$ dmesg | tail -1
[ 4112.408499] Fortune module unloaded.
sergey@sergey-VirtualBox:~/lab4/fort$

```

Созданная поддиректория в /proc:

```

dr-xr-xr-x  2 root      root           0 map 28 16:28 Dir_in_proc

```

Созданная символическая ссылка:

```

lrwxrwxrwx  1 root      root           13 map 28 16:28 Symbolic_in_proc -> /proc/fortune

```

Созданный файл:

```

-rw-rw-rw-  1 root      root           0 map 28 16:36 fortune

```