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:

```
#include <stdio.h>
#define BUFFSIZE 0x1000

int main(int argc, char* argv[])
{
    char buffer[BUFFSIZE];
    int len;
    int i;
    FILE* f;

     f = fopen("/proc/self/environ","r");
     while ((len = fread(buffer, 1, BUFFSIZE, f)) > 0)
     {
         for (i = 0; i < len; i++)
             if (buffer[i] == 0)
                 buffer[i] = 10;
         buffer[len - 1] = 10;

```

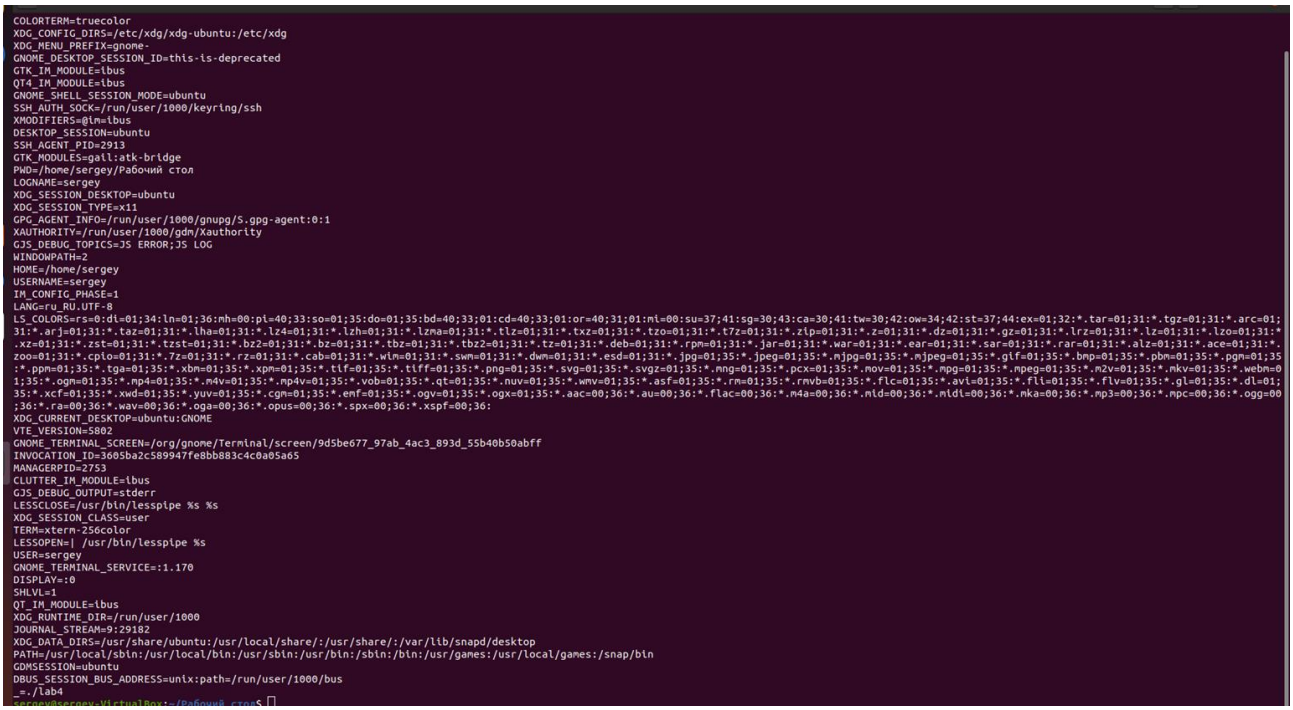

Рис.1 Вывод информации об окружении процесса

Содержимое файла /proc/[pid]/stat:

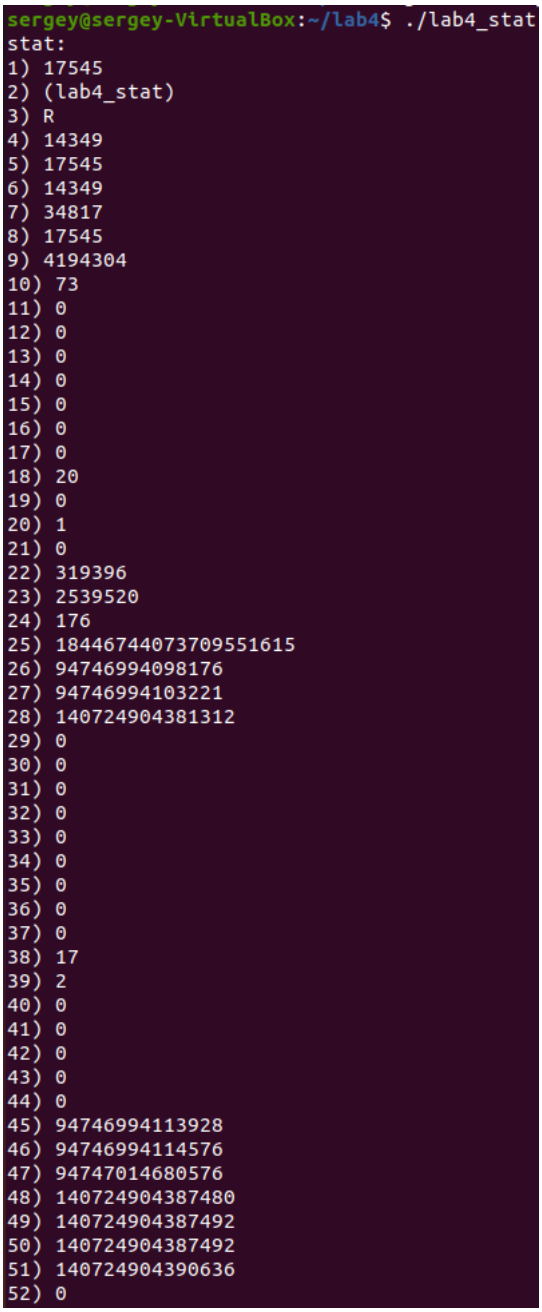

Рис.2 Вывод информации, характеризующей состояние процесса

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 мигов 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) vsize - 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) cnswap - 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:

```
#include <stdio.h>
#include <string.h>

#define BUFFSIZE 0x1000

int main(int argc, char *argv)
{
    char buffer[BUFFSIZE];
    int len;
     FILE *f;

    f = fopen("/proc/self/stat","r");
    fread(buffer, 1, BUFFSIZE, f);
    char* p_ch = strtok(buffer, " ");

    printf("stat: \n");

    while (p_ch != NULL)
    {
        printf("%s \n", p_ch);
        p_ch = strtok(NULL, " ");
    }
    fclose(f);
    return 0;
}
```

Содержимое файда 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,
    "..") != 0))
22.         {
23.             sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
24.             readlink(path, string, BUFFSIZE);
25.             path[BUFFSIZE] = '\0';
26.             printf("%s -> %s\n", dirp->d_name, string);
27.         }
28.     }
29.     closedir(dp);
30.     return 0;
31. }
```

Рис. 2 Результат работы программы, содержимое директории fd

Содержимое директории cmdline

4. Листинг программы для вывода содержимого директории cmdline.

```c
#include <stdio.h>
#include <unistd.h>

#define BUFFSIZE 0x1000

int main(int argc, char *argv)
{
    char buffer[BUFFSIZE];
    FILE *f;
     int len;

    f = fopen("/proc/self/cmdline", "r");
    len = fread(buffer, 1, BUFFSIZE, f);
    buffer[--len] = 0;

    printf("cmdline for %d \nprocess = %s\n", getpid(), buffer);
    fclose(f);

    return 0;
}
```

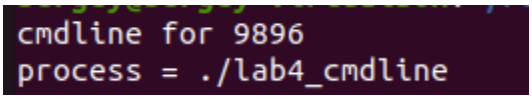

Рис. 3 Результат работы программы, содержимое директории cmdline

Часть II

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.

Листинг программы:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/vmalloc.h>
#include <asm/uaccess.h>
#include <linux/uaccess.h>

#define COOKIE_BUF_SIZE PAGE_SIZE


MODULE LICENSE("GPL");
MODULE AUTHOR("Sergey Mirzoyan");

ssize_t fortune_read(struct file *file, char *buf, size_t count, loff_t *f_pos);
ssize_t fortune_write(struct file *file, const char *buf, size_t count, loff_t *f_pos);
int fortune_init(void);
void fortune_exit(void);

struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = fortune_read,
    .write = fortune_write,
};

```

```
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 мар 28 16:28 Dir_in_proc
```

Созданная символьная ссылка:

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

Созданный файл:

```
-rw-rw-rw-  1 root            root                          0 мар 28 16:36 fortune
```