



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: Гасанзаде М.А.,

Группа: ИУ7-66Б

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`.
- `XDGMENU_PREFIX`, `XDGVNMR`, `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;

        printf("%s", buffer);
    }

    fclose(f);
    return 0;
}
```



```
0
0
0
0
0
0
17
1
0
0
0
0
0
0
94385604001256
94385604001888
94385618092032
140731883869032
140731883869036
140731883869036
140731883872244
0
```

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 SIGALRM 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) cnsvar - суммарное nswar 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 файла fd

3. Листинг программы для вывода содержимого директории fd.

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

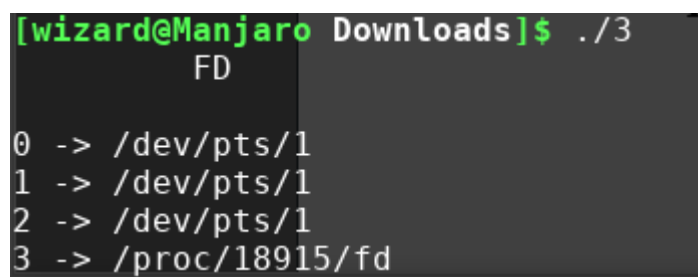
#define BUFFSIZE 0x1000

int main(int argc, char* argv)
{
    struct dirent *dirp;
    DIR *dp;

    char string[BUFFSIZE];
    char path[BUFFSIZE];

    dp = opendir("/proc/self/fd");
    puts("\t FD \n ");

    while ((dirp = readdir(dp)) != NULL)
    {
        if ((strcmp(dirp->d_name, ".") != 0 ) && (strcmp(dirp->d_name, "..")
!= 0))
        {
            sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
            readlink(path, string, BUFFSIZE);
            path[BUFFSIZE] = '\0';
            printf("%s -> %s\n", dirp->d_name, string);
        }
    }
    closedir(dp);
    return 0;
}
```



```
[wizard@Manjaro Downloads]$ ./3
FD
0 -> /dev/pts/1
1 -> /dev/pts/1
2 -> /dev/pts/1
3 -> /proc/18915/fd
```

Рис2. Результат работы программы, содержимое директории fd

Содержимое директории cmdline

4. Листинг программы для вывода содержимого директории cmdline.

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

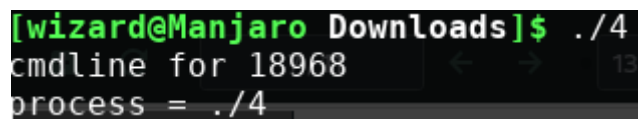
#define BUFFSIZE 0x1000

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;
}
```

A terminal window with a dark background. The prompt is [wizard@Manjaro Downloads]\$ and the command ./4 is entered. The output shows the process ID 18968 and the directory path ./4.

```
[wizard@Manjaro Downloads]$ ./4
cmdline for 18968
process = ./4
```

Рис3. 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.

5. Листинг программы загружаемого модуля ядра

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

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Message/text Input/Output Kernel Module");
MODULE_AUTHOR("Hasanzade M.A.");

#define MAX_LENGTH PAGE_SIZE

static char* storage;
static int index;
static int next;
static struct proc_dir_entry *proc_entry;
static struct proc_dir_entry *proc_slink;
static struct proc_dir_entry *proc_dir;

ssize_t storage_write(struct file *filp, const char __user *buff,
                     size_t len, loff_t *data) {
    int space_available = (MAX_LENGTH-index)+1;

    if (len > space_available) {
        printk(KERN_INFO "storage: Storage is full!\n");
        return -ENOSPC;
    }

    if (copy_from_user(&storage[index], buff, len))
        return -ENOSPC;

    index += len;
    storage[index-1] = 0;
    return len;
}

ssize_t storage_read(struct file *filp, char __user *buffer, size_t count,
                    loff_t *off) {
    int len;

    if (*off > 0 || index == 0)
        return 0;

    if (next >= index)
        next = 0;

    len = copy_to_user(buffer, &storage[next], count);
    next += len;
    *off += len;
    return len;
}
```

```

static const struct file_operations proc_file_fops = {
    .owner = THIS_MODULE,
    .write = storage_write,
    .read = storage_read,
};

static int __init my_module_init(void) {
    int ret = 0;

    storage = (char *)vmalloc(MAX_LENGTH);

    if (!storage)
        ret = -ENOMEM;
    else {
        memset(storage, 0, MAX_LENGTH);
        proc_dir = proc_mkdir("directory", NULL);
        proc_entry = proc_create("storage", 0644, proc_dir,
&proc_file_fops);
        proc_slink = proc_symlink("sym_to_storage", proc_dir,
"/proc/directory/storage");

        if (proc_entry == NULL) {
            ret = -ENOMEM;
            vfree(storage);
            printk(KERN_INFO "storage: Couldn't create proc entry\n");
        }
        else {
            index = 0;
            next = 0;
            printk(KERN_INFO "storage: Module loaded.\n");
        }
    }

    return ret;
}

static void __exit my_module_cleanup(void) {
    remove_proc_entry("storage", NULL);
    remove_proc_entry("directory", NULL);
    remove_proc_entry("sym_to_storage", NULL);
    if (storage)
        vfree(storage);

    printk(KERN_INFO "storage: Module exited.\n");
}

module_init(my_module_init);
module_exit(my_module_cleanup);

```

Результат работы программы:

```

ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ lsmod | grep mykernel
mykernel                16384  0

```

```

ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ echo "Hello" > /proc/directory/storage
ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ echo "World!" > /proc/directory/storage
ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ cat /proc/directory/storage
HelloWorld!ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ cat /proc/directory/storage

```

Созданная поддиректория в /proc:

```
-r--r--r-- 1 root root 0 anp 4 15:42 devices
dr-xr-xr-x 2 root root 0 anp 4 15:42 directory
dr-xr-xr-x 2 root root 0 anp 4 15:42 dir_in_proc
```

Результат команды `ls -al`:

```
ztsugumi@ztsugumi-VirtualBox:~/Desktop/test$ ls -al /proc/directory/
total 0
dr-xr-xr-x  2 root root  0 anp  4 15:42 .
dr-xr-xr-x 244 root root  0 anp  1 17:29 ..
-rw-rw-rw-  1 root root  0 anp  4 15:43 storage
lrwxrwxrwx  1 root root 23 anp  4 15:43 sym_to_storage -> /proc/directory/storage
```