



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»

ОТЧЕТ

к лабораторной работе №4

По курсу: «Операционные системы»

На тему: «Файловая система /proc»

Студентка ИУ7-65Б
Оберган Т.М

Преподаватель
Рязанова Н.Ю.

2020 г.

Оглавление

Часть 1	3
/proc/[pid]/cmdline	3
Листинг программы:	3
Результат работы программы:.....	3
/proc/[pid]/environ.....	4
Листинг программы:	4
Результат работы программы:.....	4
/proc/[pid]/stat	6
Листинг программы:	6
Результат работы программы:.....	9
/proc/[pid]/fd/	10
Листинг программы:	10
Результат работы программы:.....	11
Часть 2	11
Загружаемый модуль ядра.....	11
Листинг программы:	11
Результат работы программы:.....	13

Часть 1

/proc/[pid]/cmdline

Листинг программы:

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

int main(int argc, char* argv[])
{
    char buf[BUFFSIZE];
    int len;
    FILE* f = fopen("/proc/self/cmdline", "r");

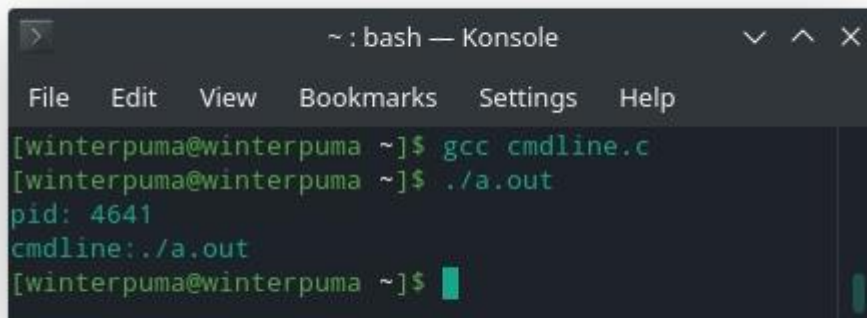
    len = fread(buf, 1, BUFFSIZE, f);
    buf[len-1] = 0;

    printf("pid: %d\ncmdline:%s\n", getpid(), buf);

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

Результат работы программы:

Содержит полную командную строку процесса. В случае, если процесс находится в состоянии зомби, файл пуст.



```
~ : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma ~]$ gcc cmdline.c
[winterpuma@winterpuma ~]$ ./a.out
pid: 4641
cmdline:./a.out
[winterpuma@winterpuma ~]$
```

/proc/[pid]/environ

Листинг программы:

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

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

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

        buf[len - 1] = 10;
        printf("%s", buf);
    }
    fclose(f);
    return 0;
}
```

Окружение — это набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, доступный каждому пользовательскому процессу. Иными словами, окружение — это набор переменных окружения.

Результат работы программы:

[illegible]

Данный файл содержит исходное окружение, которое было установлено при запуске текущего процесса (вызове `execlve()`).

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.

/proc/[pid]/stat

Листинг программы:

```
#include <stdio.h>
#include <string.h>
#define BUFSIZE 0x1000

int main(int argc, char*argv)
{
    char buf[BUFSIZE];
    int n = 0;
    FILE *f;
    f = fopen("/proc/self/stat", "r");
    fread(buf, 1, BUFSIZE, f);
    char* p_ch = strtok(buf, " ");

    printf("\n stat: \n");

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

Содержимое файла /proc/[pid]/stat:

- 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) smajflt - 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) `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()`.

Результат работы программы:

```
~ : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma ~]$ ./a.out

stat:
1. 4357
2. (a.out)
3. R
4. 3171
5. 4357
6. 3171
7. 34818
8. 4357
9. 4194304
10. 74
11. 0
12. 0
13. 0
14. 0
15. 0
16. 0
17. 0
18. 20
19. 0
20. 1
21. 0
22. 472827
23. 2359296
24. 127
25. 18446744073709551615
26. 94680123023360
27. 94680123028277
28. 140733935473200
29. 0
30. 0
31. 0
32. 0
33. 0
34. 0
35. 0
36. 0
37. 0
38. 17
39. 7
40. 0
41. 0
42. 0
43. 0
44. 0
45. 94680123039208
46. 94680123039840
47. 94680145211392
48. 140733935481755
49. 140733935481763
50. 140733935481763
51. 140733935484912
52. 0
```

/proc/[pid]/fd/

Листинг программы:

```
#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;

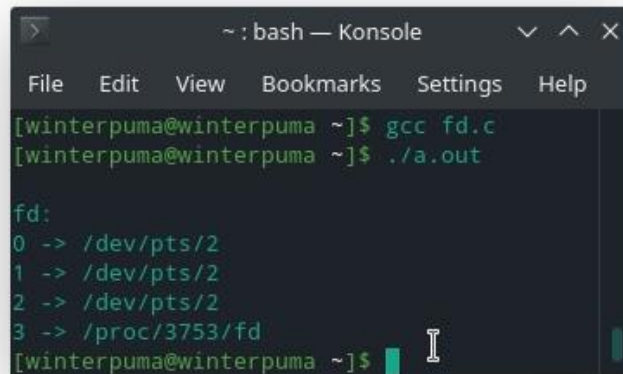
    char string[BUFFSIZE];
    char path[BUFFSIZE];
    dp = opendir("/proc/self/fd"); // open directory

    printf("\nfd:\n");

    while((dirp = readdir(dp)) != NULL) // read directory
    {
        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;
}
```

Данная поддиректория содержит одну запись для каждого файла, который открыт процессом. Имя каждой такой записи соответствует номеру файлового дескриптора и является символьной ссылкой на реальный файл.

Результат работы программы:



```
[winterpuma@winterpuma ~]$ gcc fd.c
[winterpuma@winterpuma ~]$ ./a.out

fd:
0 -> /dev/pts/2
1 -> /dev/pts/2
2 -> /dev/pts/2
3 -> /proc/3753/fd
[winterpuma@winterpuma ~]$
```

Часть 2

Загружаемый модуль ядра

Написать загружаемый модуль ядра, создать файл в файловой системе проcs, 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:

```
#include <linux/module.h>
#include <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("Dual BSD/GPL");
MODULE_AUTHOR("Obergan T.M.");

int fortune_init(void);
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);
void fortune_exit(void);

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

```

char *cookie_buf;
struct proc_dir_entry *proc_file;
unsigned int read_index;
unsigned int write_index;

int fortune_init(void)
{
    cookie_buf = vmalloc(COOKIE_BUF_SIZE);

    if (!cookie_buf)
    {
        printk(KERN_INFO "Error: can't malloc cookie buffer\n");
        return -ENOMEM;
    }

    memset(cookie_buf, 0, COOKIE_BUF_SIZE);
    proc_file = proc_create("fortune", 0666, NULL, &fops);

    if (!proc_file)
    {
        vfree(cookie_buf);
        printk(KERN_INFO "Error: can't create fortune file\n");
        return -ENOMEM;
    }

    read_index = 0;
    write_index = 0;

    proc_mkdir("Dir_fort", NULL);
    proc_symlink("Symbolic_fort", NULL, "/proc/fortune");

    printk(KERN_INFO "Fortune module loaded successfully\n");
    return 0;
}

ssize_t fortune_read(struct file *file, char *buf, size_t count, loff_t *f_pos)
{
    int len;

    if (write_index == 0 || *f_pos > 0)
        return 0;

    if (read_index >= write_index)
        read_index = 0;

    len = sprintf(buf, "%s\n", &cookie_buf[read_index]);
    read_index += len;
    *f_pos += len;

    return len;
}

ssize_t fortune_write(struct file *file, const char *buf, size_t count, loff_t *f_pos)
{
    int free_space = (COOKIE_BUF_SIZE - write_index) + 1;

    if (count > free_space)
    {
        printk(KERN_INFO "Error: cookie pot is full\n");
        return -ENOSPC;
    }
}

```

```

    if (copy_from_user(&cookie_buf[write_index], buf, count))
        return -EFAULT;

    write_index += count;
    cookie_buf[write_index-1] = 0;

    return count;
}

void fortune_exit(void)
{
    remove_proc_entry("fortune", NULL);

    if (cookie_buf)
        vfree(cookie_buf);

    printk(KERN_INFO "Fortune module unloaded\n");
}

module_init(fortune_init);
module_exit(fortune_exit);

```

Результат работы программы:

```

[winterpuma@winterpuma ~]$ sudo insmod fortune.ko
[winterpuma@winterpuma ~]$ dmesg | tail -1
[ 3330.463198] Fortune module loaded successfully
[winterpuma@winterpuma ~]$ echo "Doing OS lab4" > /proc/fortune
[winterpuma@winterpuma ~]$ cat /proc/fortune
Doing OS lab4
[winterpuma@winterpuma ~]$ sudo rmmod fortune
[winterpuma@winterpuma ~]$ dmesg | tail -1
[ 3460.391494] Fortune module unloaded

```

Созданный файл:

```
-rw-rw-rw- 1 winterpuma winterpuma 0 Apr 2 22:10 fortune
```

Символьная ссылка:

```
lrwxrwxrwx 1 winterpuma winterpuma 13 Apr 2 22:10 Symbolic_fort -> /proc/fortune
```

Поддиректория в /proc:

```
dr-xr-xr-x 2 winterpuma winterpuma 0 Apr 2 22:10 Dir_fort
```