|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

|  |  |
| --- | --- |
| **Тема** Виртуальная файловая система /proc  **Студент** Сушина А.Д.  **Группа** ИУ7-61б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю, |  |

Москва.

2020 г

Оглавление

[1 Задание на лабораторную работу 3](#__RefHeading___Toc853_2281104548)

[2 Листинг программ 3](#__RefHeading___Toc867_2281104548)

[3 Результаты работы программы 7](#__RefHeading___Toc857_2281104548)

# Задание на лабораторную работу

**Цель лабораторной работы**: изучить работу файловой системы /proc.

**Часть 1.**

* В пользовательском режиме вывести на экран информацию об окружении процесса с комментариями;
* В пользовательском режиме вывести на экран информацию о состоянии процесса с комментариями;
* Вывести информацию из файла cmdline и директории fd.

**Часть 2.**

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. Продемонстрировать это.

# Листинг программ

На листингах 1-2 представлен код программ для 1 и второй части лабораторной работы.

|  |
| --- |
| Листинг 1. Часть 1. Вывод информации об окружении, состоянии процесса и информацию из cmdline и директории fd.   1. static int i = 0; 2. void **statOutput**(char \*buf) 3. { 4. int len = strlen(buf); 5. int currentName = 0; 6. char \*pch = strtok(buf, " "); 8. while (pch != NULL && i < 51) 9. { 10. printf("\n%15s:\t %s", outputNames[i], pch); 11. pch = strtok(NULL, " "); 12. i++; 13. } 14. } 15. void **simpleOutput**(char \*buf) 16. { 17. printf("%s\n", buf); 18. } 19. void **read\_one\_file**(char\* filename, void (\*print\_func)(char\*)) 20. { 21. char buf[BUF\_SIZE]; 22. int i, len; 23. FILE \*f = fopen(filename, "r"); 24. while ((len = fread(buf, 1, BUF\_SIZE, f)) > 0) 25. { 26. for (i = 0; i < len; i++) 27. if( buf[i] == 0) 28. buf[i] = 10; 29. buf[len - 1] = 0; 30. print\_func(buf); 31. } 32. fclose(f); 33. } 34. int **main**(int argc, char \*argv[]) 35. { 36. printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"); 37. printf("STAT \n\n"); 38. read\_one\_file("/proc/self/stat", statOutput); 39. printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"); 40. printf("ENVIRON\n\n"); 41. read\_one\_file("/proc/self/environ", simpleOutput); 42. printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"); 43. printf("CMDLINE\n\n"); 44. read\_one\_file("/proc/self/cmdline", simpleOutput); 45. printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"); 46. printf("FD\n\n"); 47. execl("/bin/ls", "ls", "/proc/self/fd", NULL); 48. return 0; 49. } |
| Листинг 2. Часть 2. Модуль ядра fortune   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 <linux/uaccess.h> 8. #include <linux/sched.h> 9. #include <linux/init\_task.h> 10. MODULE\_LICENSE("GPL"); 11. MODULE\_DESCRIPTION("Fortune Cookie Kernel Module"); 12. #define COOKIE\_BUF\_SIZE PAGE\_SIZE 13. #define TEMP\_BUF\_SIZE 256 14. ssize\_t **fortune\_read**(struct file \*file, char \*buf, size\_t count, loff\_t \*f\_pos); 15. ssize\_t **fortune\_write**(struct file \*file, const char \*buf, size\_t count, loff\_t \*f\_pos); 16. int **fortune\_init**(void); 17. void **fortune\_exit**(void); 18. struct file\_operations fops = { 19. .owner = THIS\_MODULE, 20. .read = fortune\_read, 21. .write = fortune\_write, 22. }; 23. static char \*cookie\_buf; 24. static struct proc\_dir\_entry \*proc\_entry; 25. static unsigned read\_index; 26. static unsigned write\_index; 27. char temp[TEMP\_BUF\_SIZE]; 28. struct task\_struct \*task = &init\_task; 29. int len; 30. ssize\_t **fortune\_read**(struct file \*file, char \*buf, size\_t count, loff\_t \*f\_pos) 31. { 32. if (\*f\_pos > 0) 33. return 0; 34. if (read\_index >= write\_index) 35. read\_index = 0; 36. len = 0; 37. if (write\_index > 0) 38. { 39. len = sprintf(temp, "%s\n", &cookie\_buf[read\_index]); 40. copy\_to\_user(buf, temp, len); 41. buf += len; 42. read\_index += len; 43. } 44. \*f\_pos += len; 45. return len; 46. } 47. ssize\_t **fortune\_write**(struct file \*file, const char \*buf, size\_t count, loff\_t \*f\_pos) 48. { 49. int space\_available = (COOKIE\_BUF\_SIZE - write\_index) + 1; 50. if (count > space\_available) 51. { 52. printk(KERN\_INFO "+\_+ cookie pot is full\n"); 53. return -ENOSPC; 54. } 55. if (copy\_from\_user(&cookie\_buf[write\_index], buf, count)) 56. return -EFAULT; 57. write\_index += count; 58. cookie\_buf[write\_index - 1] = 0; 59. return count; 60. } 61. int **fortune\_init**(void) 62. { 63. cookie\_buf = (char \*) vmalloc(COOKIE\_BUF\_SIZE); 64. if (!cookie\_buf) 65. { 66. printk(KERN\_INFO "+\_+ not enough memory for the cookie pot\n"); 67. return -ENOMEM; 68. } 69. memset(cookie\_buf, 0, COOKIE\_BUF\_SIZE); 70. proc\_entry = proc\_create("fortune", 0666, NULL, &fops); 71. if (!proc\_entry) 72. { 73. vfree(cookie\_buf); 74. printk(KERN\_INFO "+\_+ Couldn't create proc entry\n"); 75. return -ENOMEM; 76. } 77. read\_index = 0; 78. write\_index = 0; 79. proc\_mkdir("my\_dir\_in\_proc", NULL); 80. proc\_symlink("my\_symbolic\_in\_proc", NULL, "/proc/fortune"); 81. printk(KERN\_INFO "+\_+ fortune module loaded.\n"); 82. return 0; 83. } 84. void **fortune\_exit**(void) 85. { 86. remove\_proc\_entry("fortune", NULL); 87. if (cookie\_buf) 88. vfree(cookie\_buf); 89. printk(KERN\_INFO "+\_+ fortune module unloaded.\n"); 90. } 91. module\_init(fortune\_init); 92. module\_exit(fortune\_exit); |

# Результаты работы программы

|  |
| --- |
| Рис 1. Вывод программы из части 1. Содержание файла STAT. |
| Рис 2. Вывод программы из части 1. Содержимое файла ENVIRON |
| Рис 3. Вывод программы из части 1. Содержимое файла Environ, cmdline. |
| Рис 4. Вывод программы из части 1. Содержимое директории fd. |
| Рис 5. Демонстрация работы загружаемого модуля ядра Fortune |

На рисунке 1 представлено содержимое файла stat. В этом файле содержится вся информация о процессе. В данном случае это процесс main.exe (написанная мной программа, которая выводит информацию из файла).

На рисунке 2 и 3 представлено содержимое файла environ.Данный файл содержит исходное окружение, которое было установлено при запуске текущего процесса. Переменные окружения разделены символами конца строки (для доступа к поддиректории текущего '\0').

Также на рисунке 3 представлен вывод файла cmdline. Данный файл содержит полную командную строку процесса, если процесс не находится в состоянии зомби, иначе файл пуст. В данном случае выводит ./main.exe, так как этой командой был запущен текущий процесс.

На рисунке 4 представлено содержимое директории fd. 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. Программа вывела 4 файла, являющиеся символическими ссылками на файлы, которые были открыты процессом.

На рисунке 5 представлен пример работы загружаемого модуля ядра fortune.

Из листинга 2 видно, что помимо файла /proc/fortune (строка 70), создается также поддериктория «my\_dir\_in\_proc» (строка 79) и символическая ссылка «my\_symbolic\_in\_proc» (строка 80). С помощью символической ссылки можно также обращаться к файлу fortune.