|  |  |
| --- | --- |
| 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*

*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.

· 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**;**

printf**(**"%s"**,** buffer**);**

**}**

fclose**(**f**);**

**return** 0**;**

**}**



**Рис1. Вывод информации об окружении процесса**

**Содержимое файла /proc/[pid]/stat:**

stat:

18879

(2)

R

18764

18879

18764

34817

18879

4194304

71

0

0

0

0

0

0

0

20

0

1

0

189028

2359296

127

18446744073709551615

94385603985408

94385603990293

140731883866128

0

0

0

0

**Вывод информации, характеризующей состояние процесса**

0

0

0

0

0

17

1

0

0

0

0

0

94385604001256

94385604001888

94385618092032

140731883869032

140731883869036

140731883869036

140731883872244

0

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 процессовпотомков, провёл в режиме пользователя.

17) cstime - количество тиков, которые процесс, ожидающий завершения процессовпотомков, 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) сnswap - 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 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**);**

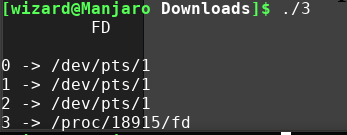
**}**

**}**

closedir**(**dp**);**

**return** 0**;**

**}**

****

**Рис2. Результат работы программы, содержимое директории fd**

**Содержимое директории cmdline**

**4. Листинг программы для вывода содержимого директории cmdline.**

**Рис3. Результат работы программы, содержимое 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**;**

**}**

**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.

**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"**);**

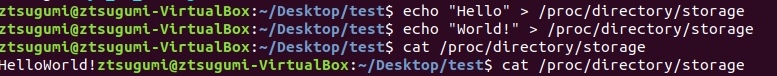
**}**

module\_init**(**my\_module\_init**);**

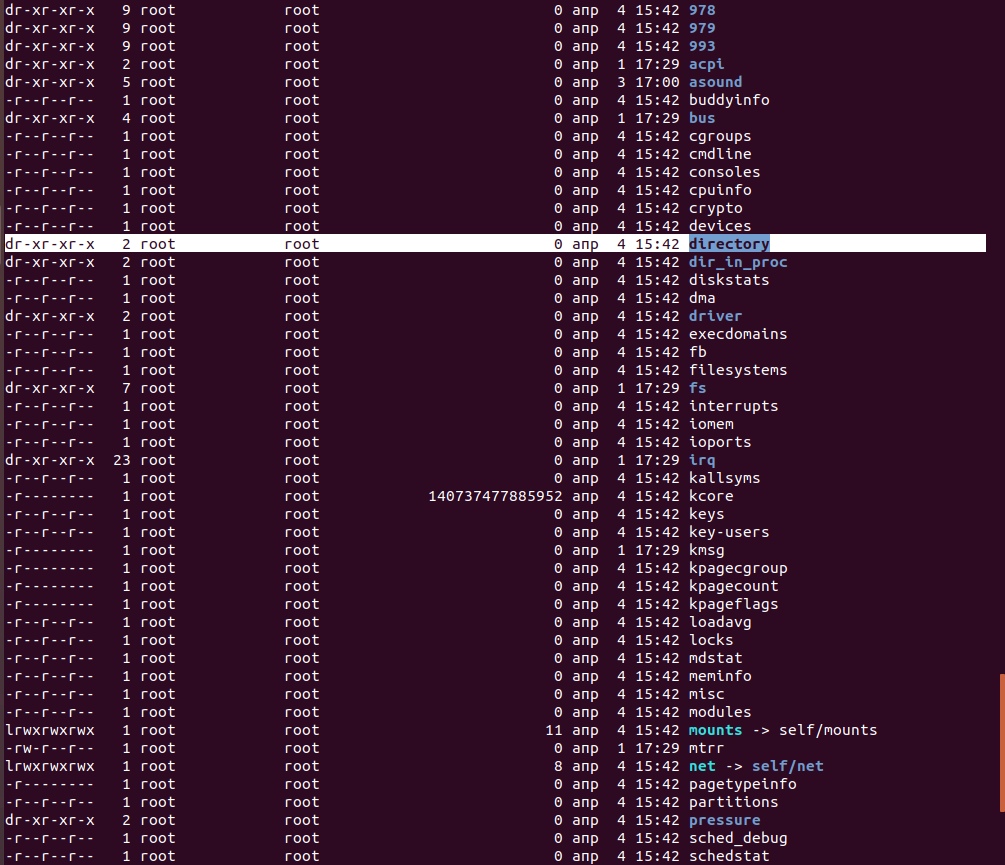
module\_exit**(**my\_module\_cleanup**);**

**Результат работы программы:**

****



**Созданная поддиректория в /proc:**



**Результат команды** ls -al:

