|  |  |  |
| --- | --- | --- |
| **Цилюрик О.И. Модули ядра Linux** | | |
| [Назад](http://rus-linux.net/MyLDP/BOOKS/Moduli-yadra-Linux/05/kern-mod-05-08.html) | **Внешние интерфейсы модуля** | [Вперед](http://rus-linux.net/MyLDP/BOOKS/Moduli-yadra-Linux/05/kern-mod-05-10.html) |

***Интерфейс /proc***

Интерфейс к файловым именам /proc (procfs) и более поздний интерфейс к именам /sys (sysfs) рассматривается как канал передачи диагностической (из) и управляющей (в) информации для модуля. Такой способ взаимодействия с модулем может полностью заменить средства вызова ioctl() для устройств, который устаревший и считается опасным. В настоящее время сложилась тенденция многие управляющие функции переносить их /proc в /sys, отображения путевых имён модулем в эти две подсистемы по своему назначению и возможностям являются очень подобными. Содержимое имён-псевдофайлов в обоих системах является только **текстовым**отображением некоторых внутренних данных ядра. Но нужно иметь в виду и ряд отличий между ними:

* Файловая система /proc является общей, «родовой» принадлежностью всех UNIX систем (Free/Open/Net BSD, Solaris, QNX, MINIX 3, ...), её наличие и общие принципы использования оговариваются стандартом POSIX 2; а файловая система /sys является сугубо Linux «изоретением» и используется только этой системой.
* Так сложилось по традиции, что немногочисленные диагностические файлы в /proc содержат зачастую большие таблицы текстовой информации, в то время, как в /sys создаётся много больше по числу имён, но каждое из них даёт только информацию об ограниченном значении, часто соответствующем одной элементарной переменной языка C: int, long, ...

Сравним:

**$ cat /proc/cpuinfo**

processor : 0

vendor\_id : GenuineIntel

cpu family : 6

model : 14

model name : Genuine Intel(R) CPU T2300 @ 1.66GHz

stepping : 8

cpu MHz : 1000.000

...

**$ wc -l cpuinfo**

58 cpuinfo

- это 58 строк текста. А вот образец информации (выбранной достаточно наугад) системы /sys:

**$ tree /sys/module/cpufreq**

/sys/module/cpufreq

└── parameters

├── debug

└── debug\_ratelimit

1 directory, 2 files

**$ cat /sys/module/cpufreq/parameters/debug**

0

**$ cat /sys/module/cpufreq/parameters/debug\_ratelimit**

1

Различия в форматном представлении информации, часто используемой в той или иной файловой системе, породили заблуждение (мне приходилось не раз это слышать), что интерфейс в /proc создаётся только для чтения, а интерфейс /sys для чтения и записи. Это совершенно неверно, оба интерфейса допускают и чтение и запись.

Теперь, когда мы кратно пробежались на качественном уровне по свойствам интерфейсов, можно перейти к примерам кода модулей, реализующих первый из этих интерфейсов. Интерфейс /proc рассматривается на примерах из архива proc.tgz. Мы будем собирать несколько однотипных модулей, поэтому общую часть определений снесём в отдельный файл:

***mod\_proc.h :***

#include <linux/module.h>

#include <linux/proc\_fs.h>

#include <linux/stat.h>

#include <asm/uaccess.h>

MODULE\_LICENSE( "GPL" );

MODULE\_AUTHOR( "Oleg Tsiliuric <olej@front.ru>" );

static int \_\_init proc\_init( void ); // предварительные определения

static void \_\_exit proc\_exit( void );

module\_init( proc\_init );

module\_exit( proc\_exit );

#define NAME\_DIR "mod\_dir"

#define NAME\_NODE "mod\_node"

#define LEN\_MSG 160 // длина буфера и сам буфер обмена

static char buf\_msg[ LEN\_MSG + 1 ] = "Hello from module!";

Файл сборки общий для всех модулей :

***Makefile :***

CURRENT = $(shell uname -r)

KDIR = /lib/modules/$(CURRENT)/build

PWD = $(shell pwd)

DEST = /lib/modules/$(CURRENT)/misc

EXTRA\_CFLAGS += -std=gnu99

TARGET1 = mod\_procr

TARGET2 = mod\_procr2

TARGET3 = mod\_proc

TARGET4 = mod\_proct

obj-m := $(TARGET1).o $(TARGET2).o $(TARGET3).o $(TARGET4).o

default:

$(MAKE) -C $(KDIR) M=$(PWD) modules

...

Основную работу по созданию и уничтожению имени в /proc выполняет пара вызовов (<linux/proc\_fs.h>):

struct proc\_dir\_entry \*create\_proc\_entry( const char \*name, mode\_t mode,

struct proc\_dir\_entry \*parent );

void remove\_proc\_entry( const char \*name, struct proc\_dir\_entry \*parent );

В результате создаётся изрядно сложная структура, в которой нас могут интересовать, в первую очередь, поля:

struct proc\_dir\_entry {

...

const char \*name;

mode\_t mode;

...

uid\_t uid;

gid\_t gid;

...

const struct file\_operations \*proc\_fops;

...

read\_proc\_t \*read\_proc;

write\_proc\_t \*write\_proc;

...

};

Смысл всех этих полей станет понятным без объяснений из рассмотрения примеров построения модулей.

Первый пример (архив proc.tgz) показывает создание интерфейса к модулю в /proc доступного только для чтения из пользовательских программ (наиболее частый случай):

***mod\_procr.c :***

#include "mod\_proc.h" // в точности списан прототип read\_proc\_t из <linux/proc\_fs.h> :

ssize\_t proc\_node\_read( char \*buffer, char \*\*start, off\_t off,

int count, int \*eof, void \*data ) {

static int offset = 0, i;

printk( KERN\_INFO "read: %d\n", count );

for( i = 0; offset <= LEN\_MSG && '\0' != buf\_msg[ offset ]; offset++, i++ )

\*( buffer + i ) = buf\_msg[ offset ]; // buffer не в пространстве пользователя!

\*( buffer + i ) = '\n'; // дополним переводом строки

i++;

if( offset >= LEN\_MSG || '\0' == buf\_msg[ offset ] ) {

offset = 0;

\*eof = 1; // возвращаем признак EOF

}

else \*eof = 0;

printk( KERN\_INFO "return bytes: %d\n", i );

if( \*eof != 0 ) printk( KERN\_INFO "EOF\n" );

return i;

};

// в литературе утверждается, что для /proc нет API записи, аналогично API чтения,

// но сейчас в <linux/proc\_fs.h> есть описание типа (аналогичного типу read\_proc\_t)

// typedef int (write\_proc\_t)( struct file \*file, const char \_\_user \*buffer,

// unsigned long count, void \*data );

static int \_\_init proc\_init( void ) {

int ret;

struct proc\_dir\_entry \*own\_proc\_node;

own\_proc\_node = create\_proc\_entry( NAME\_NODE, S\_IFREG | S\_IRUGO | S\_IWUGO, NULL );

if( NULL == own\_proc\_node ) {

ret = -ENOMEM;

printk( KERN\_ERR "can't create /proc/%s\n", NAME\_NODE );

goto err\_node;

}

own\_proc\_node->uid = 0;

own\_proc\_node->gid = 0;

own\_proc\_node->read\_proc = proc\_node\_read;

printk( KERN\_INFO "module : success!\n");

return 0;

err\_node: // обычная для модулей практика использования goto по ошибке

return ret; }

static void \_\_exit proc\_exit( void ) {

remove\_proc\_entry( NAME\_NODE, NULL );

printk( KERN\_INFO "/proc/%s removed\n", NAME\_NODE );

}

Здесь и далее, флаги прав доступа к файлу вида S\_I\* - ищите и заимствуйте в <linux/stat.h>.

Испытания:

**$ make**

...

**$ sudo insmod ./mod\_procr.ko**

**$ dmesg | tail -n1**

module : success!

**$ ls -l /proc/mod\_\***

-r--r--r-- 1 root root 0 Мар 26 18:14 /proc/mod\_node

**$ cat /proc/mod\_node**

Hello from module!

**$ dmesg | tail -n7**

module : success!

read: 3072

return bytes: 19

EOF

read: 3072

return bytes: 19

EOF

***Примечание:****Обратите внимание на характерную длину блока чтения в этой реализации, она будет отличаться в последующих реализациях.*

Несколько последовательно выполняемых операций:

**$ cat /proc/mod\_node**

Hello from module!

**$ cat /proc/mod\_node**

Hello from module!

**$ cat /proc/mod\_node**

Hello from module!

**$ sudo rmmod mod\_procr**

**$ ls -l /proc/mod\_\***

ls: невозможно получить доступ к /proc/mod\_\*: Нет такого файла или каталога

Второй пример делает то же самое, но более простым и более описанным в литературе способом create\_proc\_read\_entry() (но этот способ просто скрывает суть происходящего, но делает в точности то же самое):

***mod\_procr2.c :***

#include "mod\_proc.h"

ssize\_t proc\_node\_read( char \*buffer, char \*\*start, off\_t off,

int count, int \*eof, void \*data ) {

// ... в точности то, что и в предыдущем случае ...

};

static int \_\_init proc\_init( void ) {

if( create\_proc\_read\_entry( NAME\_NODE, 0, NULL, proc\_node\_read, NULL ) == 0 ) {

printk( KERN\_ERR "can't create /proc/%s\n", NAME\_NODE );

return -ENOMEM;

}

printk( KERN\_INFO "module : success!\n");

return 0;

}

static void \_\_exit proc\_exit( void ) {

remove\_proc\_entry( NAME\_NODE, NULL );

printk( KERN\_INFO "/proc/%s removed\n", NAME\_NODE );

}

***Примечание****(важно!): create\_proc\_read\_entry() пример того, что API ядра, доступный программисту,****намного****шире, чем список экспортируемых имён в /proc/kallsyms или /boot/System.map-2.6.\*, это происходит за счёт множества inline определений (как и в этом случае):*

**$ cat /proc/kallsyms | grep create\_proc\_**

c0522237 T create\_proc\_entry

c0793101 T create\_proc\_profile

**$ cat /proc/kallsyms | grep create\_proc\_read\_entry**

$

*Смотрим файл определений <linux/proc\_fs.h> :*

**static inline** struct proc\_dir\_entry \*create\_proc\_read\_entry(

const char \*name, mode\_t mode, struct proc\_dir\_entry \*base,

read\_proc\_t \*read\_proc, void \* data ) {

...

}

Возвращаемся к испытаниям полученного модуля:

**$ sudo insmod ./mod\_procr2.ko**

**$ echo $?**

0

**$ cat /proc/mod\_node**

Hello from module!

**$ cat /proc/mod\_node**

Hello from module!

**$ sudo rmmod mod\_procr2**

**$ cat /proc/mod\_node**

cat: /proc/mod\_node: Нет такого файла или каталога

Третий пример показывает модуль, который создаёт имя в /proc, которое может и читаться и писаться; для этого используется не специальный вызов (типа read\_proc\_t), а структура указатели файловых операций в таблице операций (аналогично тому, как это делалось в драйверах интерфейса /dev):

***mod\_proc.c :***

#include "mod\_proc.h"

static ssize\_t node\_read( struct file \*file, char \*buf,

size\_t count, loff\_t \*ppos ) {

static int odd = 0;

printk( KERN\_INFO "read: %d\n", count );

if( 0 == odd ) {

int res = copy\_to\_user( (void\*)buf, &buf\_msg, strlen( buf\_msg ) );

odd = 1;

put\_user( '\n', buf + strlen( buf\_msg ) ); // buf — это адресное пространство пользователя

res = strlen( buf\_msg ) + 1;

printk( KERN\_INFO "return bytes : %d\n", res );

return res;

}

odd = 0;

printk( KERN\_INFO "EOF\n" );

return 0;

}

static ssize\_t node\_write( struct file \*file, const char \*buf,

size\_t count, loff\_t \*ppos ) {

int res, len = count < LEN\_MSG ? count : LEN\_MSG;

printk( KERN\_INFO "write: %d\n", count );

res = copy\_from\_user( &buf\_msg, (void\*)buf, len );

if( '\n' == buf\_msg[ len -1 ] ) buf\_msg[ len -1 ] = '\0';

else buf\_msg[ len ] = '\0';

printk( KERN\_INFO "put bytes = %d\n", len );

return len;

}

static const struct file\_operations node\_fops = {

.owner = THIS\_MODULE,

.read = node\_read,

.write = node\_write

};

static int \_\_init proc\_init( void ) {

int ret;

struct proc\_dir\_entry \*own\_proc\_node;

own\_proc\_node = create\_proc\_entry( NAME\_NODE, S\_IFREG | S\_IRUGO | S\_IWUGO, NULL );

if( NULL == own\_proc\_node ) {

ret = -ENOMEM;

printk( KERN\_ERR "can't create /proc/%s\n", NAME\_NODE );

goto err\_node;

}

own\_proc\_node->uid = 0;

own\_proc\_node->gid = 0;

own\_proc\_node->proc\_fops = &node\_fops;

printk( KERN\_INFO "module : success!\n");

return 0;

err\_node:

return ret;

}

static void \_\_exit proc\_exit( void ) {

remove\_proc\_entry( NAME\_NODE, NULL );

printk(KERN\_INFO "/proc/%s removed\n", NAME\_NODE );

}

Обратите внимание, функция чтения node\_read() в этом случае принципиально отличается от аналогичной функции с тем же именем в предыдущих примерах: не только своей реализацией, но и прототипом вызова, и тем, как она возвращает свои результаты.

Испытания того, что у нас получилось:

**$ sudo insmod ./mod\_proc.ko**

**$ ls -l /proc/mod\_\***

-rw-rw-rw- 1 root root 0 Июл 2 20:47 /proc/mod\_node

**$ dmesg | tail -n1**

module : success!

**$ cat /proc/mod\_node**

Hello from module!

**$ echo новая строка > /proc/mod\_node**

**$ cat /proc/mod\_node**

новая строка

**$ cat /proc/mod\_node**

новая строка

**$ dmesg | tail -n10**

write: 24

put bytes = 24

read: 32768

return bytes : 24

read: 32768

EOF

read: 32768

return bytes : 24

read: 32768

EOF

**$ sudo rmmod mod\_proc**

**$ cat /proc/mod\_node**

cat: /proc/mod\_node: Нет такого файла или каталога

***Примечание:****Ещё раз обратите внимание на размер блока запроса на чтение (в системном журнале), и сравните с предыдущими случаями.*

Ну а если нам захочется создать в /proc не отдельное имя, а собственную иерархию имён? Как мы наблюдаем это, например, для системного каталога:

**$ tree /proc/driver**

/proc/driver

├── nvram

├── rtc

└── snd-page-alloc

0 directories, 3 files

Пожалуйста! Для этого придётся только слегка расширить функцию инициализации предыдущего модуля (ну, и привести ему в соответствие функцию выгрузки). Таким образом, по образу и подобию, вы можете создавать иерархию произвольной сложности и любой глубины вложенности (показана только изменённая часть предыдущего примера):

***mod\_proct.c :***

...

static struct proc\_dir\_entry \*own\_proc\_dir;

static int \_\_init proc\_init( void ) {

int ret;

struct proc\_dir\_entry \*own\_proc\_node;

own\_proc\_dir = create\_proc\_entry( NAME\_DIR, S\_IFDIR | S\_IRWXUGO, NULL );

if( NULL == own\_proc\_dir ) {

ret = -ENOMEM;

printk( KERN\_ERR "can't create /proc/%s\n", NAME\_DIR );

goto err\_dir;

}

own\_proc\_dir->uid = own\_proc\_dir->gid = 0;

own\_proc\_node = create\_proc\_entry( NAME\_NODE, S\_IFREG | S\_IRUGO | S\_IWUGO, own\_proc\_dir );

if( NULL == own\_proc\_node ) {

ret = -ENOMEM;

printk( KERN\_ERR "can't create /proc/%s\n", NAME\_NODE );

goto err\_node;

}

own\_proc\_node->uid = own\_proc\_node->gid = 0;

own\_proc\_node->proc\_fops = &node\_fops;

printk( KERN\_INFO "module : success!\n");

return 0;

err\_node:

remove\_proc\_entry( NAME\_DIR, NULL );

err\_dir:

return ret;

}

static void \_\_exit proc\_exit( void ) {

remove\_proc\_entry( NAME\_NODE, own\_proc\_dir );

remove\_proc\_entry( NAME\_DIR, NULL );

printk(KERN\_INFO "/proc/%s removed\n", NAME\_NODE );

}

***Примечание:****Здесь любопытно обратить внимание на то, с какой лёгкостью имя в /proc создаётся то как каталог, то как терминальное имя (файл), в зависимости от выбора единственного бита в флагах создания: S\_IFDIR или S\_IFREG.*

Теперь смотрим что у нас получилось:

**$ sudo insmod ./mod\_proct.ko**

**$ cat /proc/modules | grep mod\_**

mod\_proct 1454 0 - Live 0xf8722000

**$ ls -l /proc/mod\***

-r--r--r-- 1 root root 0 Июл 2 23:24 /proc/modules

/proc/mod\_dir:

итого 0

-rw-rw-rw- 1 root root 0 Июл 2 23:24 mod\_node

**$ tree /proc/mod\_dir**

/proc/mod\_dir

└── mod\_node

0 directories, 1 file

**$ cat /proc/mod\_dir/mod\_node**

Hello from module!

**$ echo 'new string' > /proc/mod\_dir/mod\_node**

**$ cat /proc/mod\_dir/mod\_node**

new string

**$ sudo rmmod mod\_proct**