seq-file

april 2014
Eva-Katharina Kunst
версия ядра 3.10
proc_create_data()

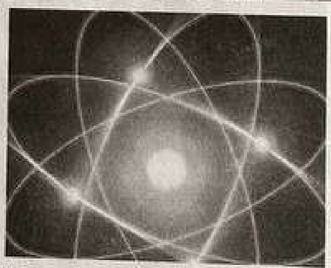
Home » Archive » 2014 » Issue 23: 10 Ti... » Kernel and driv...

Kernel and driver development for the Linux kernel

Core Technology

Article from ADMIN 23/2014

By Jürgen Quade and Eva-Katharina Kunst



Lead Image © psdesign1, Fotolia.com

The /proc filesystem facilitates the exchange of current data between the system and user. To access the data, you simply read and write to a file. This mechanism is the first step for understanding kernel programming. ü

In keeping with the central Unix philosophy that everything is a file, Linux systems publish system information through the virtual filesystems `/proc` and `/sys` (see the "Proc Filesystem" box). This brilliant mechanism gives the user read and write access to system internals with the `read()` and `write()` functions.

Proc Filesystem

Вирт. файл. сист. папки и файлы не хранит на жестком диске т.к. ядро создает их динамич. при доступе

The `/proc` virtual filesystem folders and files are not stored on a hard disk; rather, the kernel creates them dynamically on access. The term "proc" derives from "processes"; thus, it is clear that the `/proc` filesystem primarily provides information about computing processes (Figure 1).

Для каждого процесса ядро создает новую дир., имя которой дир-ид идент. проц

For each computational process, the kernel creates a new directory with the process identification number as its name. Below (внутри) the directory, you find extensive information about the job, including the call parameters (`cmdline`), shared file descriptors (`fd`) and environment variables (`environ`), and process statistics.

В добавлении к инф-ции о процессах

In addition to information about computing processes, the kernel uses the virtual filesystem to share

Вся информация о текущем CPU нах-ся в

information with the user system and receive configurations. All data for the current CPU is in /proc/cpuinfo, interrupt sources and the frequency of their occurrence is under /proc/interrupts, and info for activated device drivers with their device numbers is under /proc/devices. Writing 1 to /proc/sys/net/ipv4/ip_forward enables routing in the Linux kernel and configures the watchdog feature when written to /proc/sys/kernel/watchdog. KL:Navigating the /proc filesystem is both useful and instructive. A proc file name reveals how important a file in the /proc filesystem is. Hardware-related information is located – with the exception of CPU information – in the /sys filesystem. This is useful for driver programmers as a platform for exchanging information.

/proc $ ls

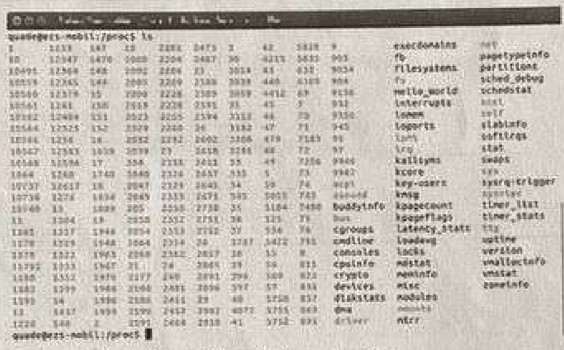


Figure 1: Directories and files in the /proc directory.

The shell, in turn, maps these access functions to the commands cat (read) and echo (write). To read the number of interrupts that have been triggered since the last boot, then, you can simply enter the cat /proc/interrupts command in a terminal. To pass network traffic through, on the other hand, the superuser only needs to write 1 to the /proc/sys/net/ipv4/ip_forward file:

```
sudo echo 1 > /proc/sys/net/ipv4/ip_forward
```

Such functions are very easy to use in scripts.

The /proc filesystem is also useful for your first steps in kernel programming: With fewer than 50 lines of code, you can make the compiler generate a module that outputs the famous "Hello World" string when accessing a proc file (Listing 1).

Listing 1

Simple Proc File

```
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>

#define PROC_FILE_NAME  "Hello_World"
static struct proc_dir_entry *proc_file;
```

```
static char *output_string;

static int prochello_show( struct seq_file *m, void *v )
{
    int error = 0;

    error = seq_printf( m, "%s\n", output_string);
    return error;
}
```

библиотека linux/seq-file.h

файл-последовательность

```
static int prochello_open(struct inode *inode, struct file *file)
{
    return single_open(file, prochello_show, NULL);
}

static const struct file_operations prochello_fops = {
    .owner  = THIS_MODULE,
    .open   = prochello_open,
    .release= single_release,
    .read   = seq_read,
};

static int __init prochello_init(void)
{
    output_string = "Hello World";
    proc_file= proc_create_data( PROC_FILE_NAME, S_IRUGO, NULL, &prochello_fops, NULL);
```

1) 2) 3) 4) 5)

структура данных которая содержит доступные функции

права доступа

определяет где в дереве каталогов будет создан файл если NULL то ниже /proc

```
    if (!proc_file)
        return -ENOMEM;
    return 0;
}

static void __exit prochello_exit(void)
{
    if( proc_file )
        remove_proc_entry( PROC_FILE_NAME, NULL );
}
```

удаляет созданный файл

```
module_init( prochello_init );
module_exit( prochello_exit );
MODULE_LICENSE("GPL");
```

Modules are known extensions of the Linux kernel and are always constructed in a similar way (Figure 2).

Чтобы управлять модулем нам нужно 2 вещи подпрограммы кот. реализуют функциональность модуля
структуру данных кот принимает адреса главных подпрограмм

To manage a module you need two things: routines that implement the functionality of the module and a data structure that receives the addresses of the main routines.

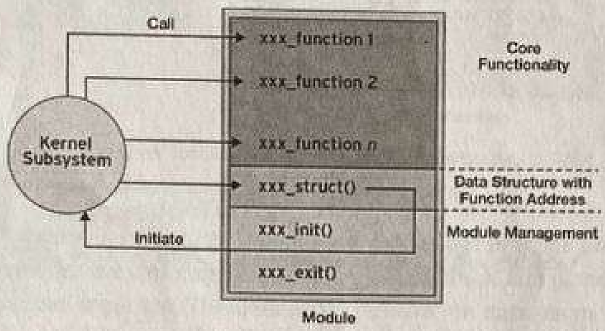


Figure 2: Kernel modules are divided into three categories.

Две управляющие функции –

The two management functions are <xxx>_init() and <xxx>_exit(), where <xxx> is the module or an arbitrary name. The <xxx>_init() function runs when the insmod command loads the module in the kernel, and <xxx>_exit() runs when the module is unloaded using rmmod.

произвольное; когда команда insmod загружает модуль в ядро; когда модуль выгружается используя rmmod

The main task of the <xxx>_init() function is to pass the data structure containing the addresses of the relevant module routines to a kernel subsystem. The module logs in to the desired subsystem(s) and itself. In this way, the kernel knows the relevant module routines and can activate them. Similarly <xxx>_exit() is for signing off.

значимые

на 1 три области типичных модулей linux ясно видно управляющие

In Listing 1, the three areas of a typical Linux module are clearly visible: the administrative functions prochello_init() and prochello_exit() (lines 29 and 38), the module routines prochello_show() and prochello_open() (lines 9 and 17), and the data structure used for logging in to the kernel prochello_fops (line 22).

регистрации

Transferring Access Rights

Передача прав доступа

Within the prochello_init() init function, the module logs in to the Linux kernel's proc subsystem by using the proc_create_data() function. It passes in the name of the proc file and the permissions for this file. Because the access rights are encoded as bit patterns, you use defines here (S_IRUGO). In the example, the owner (user), the group, and all others are given read permissions.

Она передает имя ф-ла в proc и разрешение для этого ф-ла; права на чтение

Третий параметр определяет, где в дереве директорий файловой системы /proc будет сгенерир. файл; ниже

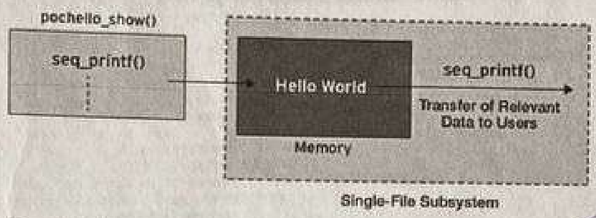
The third parameter specifies where in the directory tree of the /proc filesystem the file is to be generated. NULL creates the file below /proc. Finally, the fourth parameter transfers the data structure that contains the access functions to the new proc file. A total of three functions are needed for read access: open(), read(), and release().

выполнять; выполнить; расширенных; сложным и; промежуточный слой отказоустойчивый, чтобы

However, you only need to implement the open routine yourself (prochello_open()). If you want, you can also implement the other routines (read() and release()), but this is complicated and only useful in exceptional cases. The kernel developers have built a fault-resistant intermediate layer for this, which reduces the complexity of the data exchange to something like printf. The basic idea of the intermediate layer is that the module programmer writes all the data to a sufficiently large memory area (Figure 3).

снижает сложность обмена данными промежуточного слоя; достаточно

использовать proc-create-data(), чтобы избежать состояния „гонки"



Эта передача более сложна, чем кажется на первый взгляд, потому что пользователь может прочитать не все данные, а часть.

Дальнейший

Figure 3: The show function uses seq_printf() to w rite the data to RAM; the single-file subsystem copies the relevant to the user.

в частности

Эта передача более сложна, чем кажется

Further processing of the data – in particular, transferring data to the proc file users – is handled by the intermediate layer. This transfer is more complicated than it appears at first glance because the user might not read all the data, just a subset. The user might retrieve the data via multiple read calls.

Промежуточный слой называется „однофайловой подсистемой" и представляет собой специальную форму файла последовательности

Более того выполняет однофайловый экземпляр произвольное

The intermediate layer is called the "single-file subsystem" and is a special form of sequence file. Above all, it implements the read and the release functions (equivalent to the `close()` system call), so they can be used without changes when accessing the proc file. To create the single-file instance of the module, programmers call the `single_open()` function, which passes the address to a routine, typically called `show()`. This show routine is given the address of a memory page (in this example, the data to be output by `seq_printf()`); `seq_printf()` can be called a more or less arbitrary number of times within the show function.

Limited Single Files

Вместо этого, пространство ограничено

Single-file instances are not intended for writing large amounts of data. Instead, the space is limited to 64KB, but that's enough for most tasks. Also, `seq_printf()` monitors overflows on every call. If that were to happen, the subsystem would automatically grow the memory buffer (e.g., by creating a new one and copying) and would then write the data. If scaling is impossible, `seq_printf()` returns a negative error code. Professional programmers evaluate the return value, of course. The `single_open()` function, which receives the address of the show function, is called in the open function for the proc file (Listing 1, line 19).

получает

Figure 4 shows how you can compile the source code in Listing 1 using the Makefile (Listing 2) followed by `insmod`, which loads the generated module into the kernel, and finally `cat`, which tests the results. If you don't want to create the proc file directly below `/proc` but in a subfolder, you can use `proc_mkdir` to create a directory. The function returns a pointer to a data structure of the type `proc_dir_entry`, which represents the newly created directory.

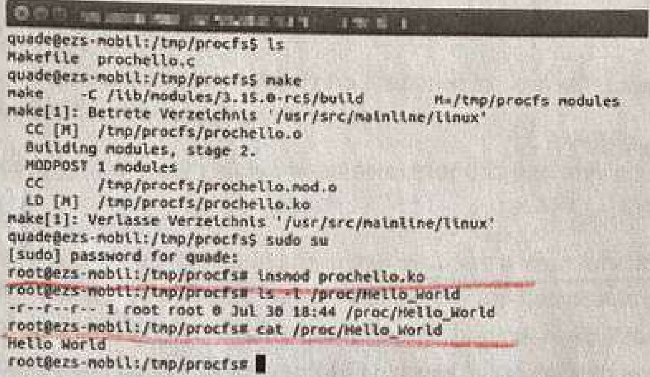


Figure 4: Generating and using the kernel module.

Listing 2

Make

```
ifneq ($(KERNELRELEASE),)
obj-m    := prochello.o

else
KDIR    := /lib/modules/$(shell uname -r)/build
PWD     := $(shell pwd)

default:
        $(MAKE) -C $(KDIR)      M=$(PWD) modules
endif
```

The `proc_mkdir()` function expects two parameters. The first contains the directory name as a string, and the second is an identifier that says where to create the new directory. If the second parameter is equal to `NULL`, the new folder is created below the `/proc` directory.

A cardinal error is to forget to delete the proc directories and files you created if you no longer need them or to remove the entire module. This is what the `remove_proc_entry()` function does. In addition to the name of the file to remove, it also supplies an identifier for the subdirectory that contains the proc file.

Hands On

The subsystem does not support the ability to transfer configuration data to the kernel by writing to a proc file. This is where the module programmer needs to lend a hand. To start, you need memory to cache the data to be written to the kernel for evaluation.

Для начала вам нужна память чтобы кэшировать данные, кот. будут записаны в ядро

If enough memory is available, you copy the data to be written from userspace to the memory buffer. You can then analyze the data and perform the action requested by the user. It is important not to transfer more data than you can buffer between userspace and kernel-space. The `min()` function ensures that

Затем вы можете анализировать данные и выполнить действия, запрошенные пользователем this does not happen.

Additionally, module programmers must constantly be prepared to deal with users trying to overwrite kernel memory by providing incorrect address information.

и только копирует допустимые адреса памяти

The transfer function `copy_from_user()` also watches out for this and only copies acceptable memory addresses.

показывает дополнение примера кот. включает ф-ю write

Listing 3 shows an extension of the proc file example that includes a write function. It writes the keyword `deutsch` to the proc file, which translates the string returned on read access to the German version: *Hallo Welt* . However, if you write `english` to the proc file, the original *Hello World* appears again the next time you read. The original code requires further changes, in addition to entering the code from Listing 3.

Listing 3

Extended Proc File

```
static ssize_t prochello_write( struct file *instanz, const char __user \
 *buffer, size_t max_bytes_to_write, loff_t *offset )
{
    ssize_t to_copy, not_copied;

    to_copy = min( max_bytes_to_write, sizeof(kernel_buffer) );

    not_copied = copy_from_user(kernel_buffer,buffer,to_copy);
    if (not_copied==0) {
        printk("kernel_buffer: \"%s\"\n", kernel_buffer);
        if (strncmp( "deutsch", kernel_buffer, 7)==0) {
            output_string = TEXT_GERMAN;
        }
        if (strncmp( "english", kernel_buffer, 7)==0) {
            output_string = TEXT_ENGLISH;
        }
    }
    return to_copy - not_copied;
}
```

First, you need to add the write function to the data structure. To do this, you add the `file_operations prochello_fops` structure to the line:

```
.write = prochello_write,
```

proc_hello_write,

To allow write access, you then need to modify the access privileges on calling `proc_create_data`: Instead of `S_IRUGO`, you need `S_IRUGO | S_IWUGO`.

Finally, four lines need to be added to the program header:

```
#include <asm/uaccess.h>
static char kernel_buffer[256];
#define TEXT_GERMAN  "Hallo Welt"
#define TEXT_ENGLISH "Hello World"
```

After making these changes and compiling, then unloading the old module version and reloading the driver, write access should be possible.

The code for implementing proc files is a good template for your own development. Essentially, you need to change the name of the proc file and the show function.

If you have extensive output that changes frequently, you will probably want to look to sequence files instead. Unfortunately, earlier writings on this topic are not really that useful as a guide (see the box "Changes in Kernel 3.10").

Changes in Kernel 3.10

Ядро 3.10 больше не поддерживает ф-ю create_proc_entry()

proc_create_data

Kernel 3.10 no longer supports the ~~create_proc_entry~~() function:

```
proc_file = create_proc_entry("example_file", S_IRUGO, proc_dir );
if (proc_file) {
    proc_file->read_proc = proc_read;
    proc_file->data = NULL;
}
```

Вместо программисты используют представленную в этой статье кот. использует

Instead, programmers use the function presented in this article, `proc_create_data()`, which uses different parameterization. Whereas individual elements of the `proc_dir_entry` data structure had to be initialized in some of the earlier versions of the kernel, developers working with a more recent kernel version reserve a structure that is already familiar from driver development, `struct file_operations`, and assign the access methods (`open()`, `read()`, `release()`):

несколько; в некоторых ранних версиях ядра разработчики работающие с более; близкий из; описываемой

```
static struct file_operations example_proc_fops = {
    .owner = THIS_MODUL,
    .open = example_proc_open,
    .read = example_proc_read,
    .release = example_proc_release,
}
[...]
static int __init example_proc_init(void)
{
  proc_file = proc_create_data(    "example_file", S_IRUGO, proc_dir,
      &example-proc_fops, NULL );
[...]
```

Более поздняя версия ядра резервирует структуру, которая уже знакома по разработке драйверов

proc_create_data()

методы отличаются от предыдущих версий.

The `read()` and `write()` access methods differ compared with previous versions. The former `peof` parameter, which used to indicate that the system had read all the data, no longer exists. In contrast, the current version writes data to the memory address passed in as a parameter and then returns the number of characters written (if the single-file subsystem is not used, unlike the description in this article).

The Author

Eva-Katharina Kunst has been an open source fan since the early days of Linux. Jürgen Quade is a professor at the Niederrhein University; he published his third Linux book *Learning Embedded Linux with the Raspberry Pi* in April 2014.