Linux 驱动之模块参数和符号导出

——编写者:草根老师(程姚根)

由于水平有限,文档中难免有错误的地方，希望大家踊跃拍砖。大家可以通过邮箱:chengyaogen@163.com告诉我，以便改正,谢谢!

一、给模块传递参数

当我们加载一个模块到Linux 内核的时候，Linux 内核允许向这个模块传递一些参数。这样设计的好处就是，让我们的模块操作起来更灵活，我们可以通过给它传递不同的参数来完成不同的功能。例如:我们写一个模块程序，来完成硬件中断的操作。在Linux 操作系统中，每个中断都由一个中断号。如果我们在模块里面将中断号写死，那我们的模块只能响应特定的中断了。如果我们把中断号作为参数传递给我们的模块，那么我们的模块就可以完成对不同的中断进行操作。

那怎么向模块传递参数呢?很简单，Linux 内核都给我们做好了，我们只需调用相应的接口就可以了。

(1)在模块里面，声明一个变量（全局变量）,用来接收用户加载模块时传递的参数

函数原型:

module_param(name,type,perm)

参数　:

@name 用来接收参数的变量名

@type 参数的数据类型

Linux 内核支持的模块参数类型	
bool	布尔值(true/false),关联的变量应该是int类型
invbool	bool的反值,例如赋值为true,实际值为false
charp	字符指针类型，内核为用户提供的字符串分配内存,并设置此指针保存其首地址
int	整型
long	长整型
short	短整型
uint	无符号整型
ulong	无符号长整型
ushort	无符号短整型

@指定参数访问权限。

每个模块的参数，最后都会表现在sysfs文件系统中，也就是说最后会在系统的/sys/module/模块名字/parameters/路径下看到以参数名命名的文件。这个文件的权限就是这里指定的权限。如果perm的值为0,则在sysfs文件系统中不会生成参数对应的文件。

典型使用案列：

我们可以在模块(test.ko)里面写如下代码，接收用户加载模块时传递的参数

```
static unsigned int var = 0;
module_param(var,uint,0400);
```

在加载模块的时候,传递参数:

```
insmod test.ko  var=100
```

最后模块里面的全局变量var 的值就为100了。

如果我要传递一个字符串到模块里面，该怎么操作呢?

```
static char *string;
module_param(string,charp,0400);
```

在加载模块的时候,传递参数:

```
insmod test.ko  string="CYG";
```

有人可能会问，这段代码是不是有bug,因为你的string 指针是一个野指针。
没关系的啦，内核会自动给用户传递的字符串分配空间的，然后用string指针保存字符串所在内存的首地址。

(2) 让模块内部变量的名字和加载模块时传递的参数名不同

函数原型:

module_param_named(name_out,name_in,type,perm);

参数　:

@name_out 在加载模块时，参数的名字

@name_in 模块内部变量的名字

@type 参数类型

@perm 访问权限

典型使用案列：

```
static  int  var  = 0;
```

module_param_named(var_out,var,int,0400);

在加载模块的时候,传递参数:

insmod test.ko var_out=100

嗯，var_out 就是模块变量var在外部的名字，此时var 的值为100

(3)加载模块的时候，传递字符串到模块的一个全局字符数组里面

函数原型:
module_param_string(name,string,len,perm);
参数　:
@name　在加载模块时，参数的名字
@string　模块内部字符数组的名字
@len　模块内部字符数组的大小
@perm　访问权限

典型使用案列：
static int buffer[LEN] ;
module_param_string(buffer_out,buffer,LEN,0400);

在加载模块的时候,传递参数:

insmod test.ko buffer_out="hello word"

嗯，加载模块的时候，内核直接把"hello word"字符串拷贝到buffer数组中。

(4)加载模块的时候，传递参数到模块的数组中

函数原型:
module_param_arry(name,type,num_point,perm);

参数　:
@name　模块的数组名，也是外部指定的参数名
@type　模块数组的数据类型
@num_point　用来获取用户在加载模块时传递的参数个数，NULL:不关心用户传递的参数个数
@perm　访问权限

典型使用案列：
static int my_arry[3];
int num;

module_param_arry(my_arry,int,&num,0400);

在加载模块的时候,传递参数:(多个参数以" , "隔开)
insmod test.ko my_arr=1,2,3

注意: 以上接口在调用的时候，perm指定的权限不能让普通用户具有写权限，否则编译会报错

---

(5)给模块里面每个接收用户参数的变量指定一个描述信息

函数原型:
MODULE_PARM_DESC(name,describe);

参数　:
@name　变量名
@describe　描述信息的字符串

终于说完了，下面我们一起来实现以下吧:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>

static int var1 = 0;
module_param(var1, int, 0644);
MODULE_PARM_DESC(var1, "Get value from user.\n");

static int var2 = 0;
module_param_named(var2_out, var2, int, 0644);
MODULE_PARM_DESC(var2, "Test var2 named var2_out.\n");

static char *string = NULL;
module_param(string,charp, 0444);
MODULE_PARM_DESC(string, "Test module param string.\n");

static char  buffer[10];
module_param_string(buffer, buffer, sizeof(buffer), 0644);
MODULE_PARM_DESC(buffer, "Test module param string buffer.\n");
```

```
static int myarry[3];
int num;
module_param_array(myarry,int,&num, 0444);
MODULE_PARM_DESC(myarry, "Test module param arry.\n");
```

```
int __init init_parm_module(void)
{
    int i = 0;

    printk("-----------------------------\n");
    printk("var1   : %d\n",var1);
    printk("var2   : %d\n",var2);
    printk("string : %s\n",string);
    printk("buffer : %s\n",buffer);

    for(i = 0;i < num;i ++)
    {
        printk("myarry[%d] : %d\n",i,myarry[i]);
    }
    printk("-----------------------------\n");

    return 0;
}

void __exit exit_parm_module(void)
{
    printk("exit parm module\n");
    return;
}

module_init(init_parm_module);
module_exit(exit_parm_module);
```

测试的结果如下:

```
cyg@ubuntuk:~/workdir/device_driver/parm_module$ sudo insmod parm_module.ko var1
=100 var2_out=200 string="CYG"  buffer="Helloword" myarry=100,200,300
cyg@ubuntuk:~/workdir/device_driver/parm_module$ dmesg
[11124.607163] -----------------------------
[11124.607168] var1   : 100
[11124.607170] var2   : 200
[11124.607171] string : CYG
[11124.607173] buffer : Helloword
[11124.607174] myarry[0] : 100
[11124.607175] myarry[1] : 200
[11124.607176] myarry[2] : 300
[11124.607177] -----------------------------
cyg@ubuntuk:~/workdir/device_driver/parm_module$ sudo rmmod parm_module
```

二、模块符号导出

(1)什么是符号？

这里的符号主要指的是全局变量和函数。

(2)为什么要导出符号?

Linux 内核采用的是以模块化形式管理内核代码。内核中的每个模块相互之间是相互独立的，也就是说A模块的全局变量和函数，B模块是无法访问的。

有些时候，我们写一些模块代码的时候，发现部分函数功能别人已经实现了，此时我们就想如果我们可以调用他们已经实现好的函数接口就好了。那如何才能做到这点呢? 符号导出了，也就是说你可以把你实现的函数接口和全局变量导出，以供其他模块使用。

在Linux 内核的世界里，如果一个模块已经以静态的方式编译进的内核，那么它导出的符号就会出现在全局的内核符号表中。在Ubuntu 14.04系统中，Linux 内核的全局符号表在/usr/src/linux-headers-3.13.0-24-generic/Module.symvers文件中存放。如果打开这个文件，可以发现里面的内容就是：

addr ---------> 符号名 ----------->模块名--------->导出符号的宏

```
0x69c97d67  ieee80211_wake_queues    net/mac80211/mac80211    EXPORT_SYMBOL
0xdff905e5  vme_slave_free  drivers/vme/vme EXPORT_SYMBOL
0x0a89eecc  rtllib_wx_get_mode  drivers/staging/rtl8192e/rtllib EXPORT_SYMBOL
0x53a4a004  bulk_sec_desc_unpack    drivers/staging/lustre/lustre/ptlrpc/ptlrpc EXPORT_SY
0xd1b1478b  cl_attr2lvb drivers/staging/lustre/lustre/obdclass/obdclass EXPORT_SYMBOL
0x3e12f0a5  dw_spi_remove_host  drivers/spi/spi-dw  EXPORT_SYMBOL_GPL
```

(3)如何导出符号?

Linux 内核给我们提供了两个宏:
EXPORT_SYMBOL(name);
EXPORT_SYMBOL_GPL(name);

上面宏定义的任一个使得给定的符号在模块外可用. _GPL 版本的宏定义只能使符号对 GPL 许可的模块可用. 符号必须在模块文件的全局部分输出，在任何函数之外，因为宏定义扩展成一个特殊用途的并被期望是全局存取的变量的声明。

(4)模块编译时，如何寻找使用的符号？

a. 在本模块中符号表中，寻找符号(函数或变量实现)
b. 在内核全局符号表中寻找
c. 在模块目录下的Module.symvers文件中寻找

如果在这三个地方都没有找到，则编译保存

三、案例演示

模块A导出全局变量global_var 和 函数add 两个符号供模块B使用。

模块A的代码:

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR ("CYG");

static int global_var = 1000;

EXPORT_SYMBOL(global_var);

static int add(int a,int b)
{
    printk("Call moduleA add!\n");
    return (a + b);
}

EXPORT_SYMBOL_GPL(add);

static int moduleA_init(void)
{
    printk("Linux moduleA ,int!\n");
    return 0;
}

static void moduleA_exit(void)
{
    printk("Hello,Linux moduleA exit!\n");
    return;
}

module_init(moduleA_init);

module_exit(moduleA_exit);
```

模块B的代码:

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR ("CYG");

extern int global_var;
extern int add(int a,int b);

static int moduleB_init(void)
{
    printk("Linux moduleB ,int!\n");
    printk("global_var = %d\n",global_var);
    printk("a + b = %d\n",add(150,100));
    return 0;
}

static void moduleB_exit(void)
{
    printk("Hello,Linux moduleB exit!\n");
    return;
}
```

```
module_init(moduleB_init);

module_exit(moduleB_exit);
```

调试步骤:
(1)编译模块A, 然后加载模块A, 在模块A编译好后，在它的当前目录会看到一个Module.symvers文件,这里存放的就是我们模块A导出的符号。
(2)将模块A编译生成的Module.symvers文件拷贝到模块B目录下，然后编译模块B,加载模块B
(3)通过dmesg查看模块打印的信息