2021年08月18日 09:58

1、前言

在Linux驱动程序编写中，使用DEVICE_ATTR宏，可以定义一个struct device_attribute设备属性，并使用sysfs的API函数，便可以在设备目录下创建出属性文件，当我们在驱动程序中实现了show和store函数后，便可以使用cat和echo命令对创建出来的设备属性文件进行读写，从而达到控制设备的功能。

2、宏DEVICE_ATTR定义

在讲解DEVICE_ATTR宏之前，先了解一些基本的结构体，首先是struct attribute结构体，其定义在include/linux/device.h中，结构体定义如下所示：

程序代码[选择]

```
struct attribute {
    const char        *name;
    umode_t            mode;
#ifdef CONFIG_DEBUG_LOCK_ALLOC
    bool            ignore_lockdep:1;
    struct lock_class_key    *key;
    struct lock_class_key    skey;
#endif
};
```

该结构体有两个重要的成员，分别是name和mode，其中name代表属性的名称，一般表示为文件名，mode代表该属性的读写权限，也就是属性文件的读写权限。

接下来要了解的结构体为struct device_attribute，该结构体的定义在include /linux/device.h，其定义如下：

程序代码[选择]

```
/* interface for exporting device attributes */
struct device_attribute {
    struct attribute    attr;
    ssize_t (*show)(struct device *dev, struct device_attribute *attr,
            char *buf);
    ssize_t (*store)(struct device *dev, struct device_attribute *attr,
             const char *buf, size_t count);
};
```

该结构体其实是struct attribute结构体的进一步封装，并提供了两个函数指针，show函数用于读取设备的属性文件，而store则是用于写设备的属性文件，当我们在Linux的驱动程序中实现了这两个函数后，便可以使用cat和echo命令对设备属性文件进行读写操作。

了解了一下基本的结构体，接下来，进一步分析宏DEVICE_ATTR的实现，在Linux内核源码中，宏DEVICE_ATTR的定义在include/linux/device.h文件中，如下：

程序代码[选择]

```
#define DEVICE_ATTR(_name, _mode, _show, _store) \
```

```
    struct device_attribute dev_attr_##_name = __ATTR(_name, _mode, _show, _store)
```

而__ATTR宏的定义在include/linux/sysfs.h文件中，如下：

程序代码[选择]

```
#define __ATTR(_name, _mode, _show, _store) {                \
    .attr = {.name = __stringify(_name),                \
         .mode = VERIFY_OCTAL_PERMISSIONS(_mode) },        \
    .show    = _show,                        \
    .store    = _store,                        \
}
```

通过上面的宏展开可以发现，其实宏DEVICE_ATTR实现的功能就是定义了一个struct device_attribute结构体变量dev_attr_name，并对里面的成员进行初始化，包括struct attribute结构体里面的name和mode成员变量，然后还有实现属性文件读写的show和store函数赋值，非常简单。

3、使用示例

接下来对宏DEVICE_ATTR使用示例进行分析，该示例在内核中将100个字节虚拟成一个设备，在驱动中实现设备属性文件的读写函数，示例如下：

首先是设备属性的定义，以及设备属性文件读写函数的实现：

程序代码[选择]

```
static ssize_t mydevice_show(struct device *dev, struct device_attribute *attr,
        char *buf)
{
    return sprintf(buf, "%s\n", mybuf);
}
static ssize_t mydevice_store(struct device *dev, struct device_attribute *attr,
         const char *buf, size_t count)
{
    sprintf(mybuf, "%s", buf);

    return count;
}
static DEVICE_ATTR(mydevice, 0644, mydevice_show, mydevice_store);
```

在上面的代码中，使用宏DEVICE_ATTR定义了一个mydevice的属性文件，而且这个属性文件的读写权限为：文件所拥有者可读写，组和其他人只能读。代码还实现了该属性文件的读写函数mydevice_show()和mydevice_store()，当在用户空间中使用cat和echo命令时，将会调用到驱动程序中实现的两个函数。

接下来是模块的加载函数，当模块加载时将会被调用，该函数的实现代码如下：

程序代码[选择] Expand

```
static int __init mydevice_init(void)
{
    int ret;
    struct device *mydevice;

    major = register_chrdev(0, "mydevice", &myfops);
    if (major < 0) {
        ret = major;
        return ret;
    }

    myclass = class_create(THIS_MODULE, "myclass");
    if (IS_ERR(myclass)) {
        ret = -EBUSY;
        goto fail;
```

函数首先调用register_chrdev()完成一个主设备号的动态申请，设备的名称为mydevice，然后调用class_create()和device_create()在sysfs中动态创建出设备所属的类myclass和mydevice设备，需要注意的是，这两个函数调用后，要对返回的结果进行错误检测，最后，使用sysfs的API函数sysfs_create_file()在sysfs中创建出设备的属性文件，完成驱动模块的加载。

接下来是该模块的卸载函数，函数的代码如下所示：

程序代码[选择]

```
static void __exit mydevice_exit(void)
{
    device_destroy(myclass, MKDEV(major, 0));
    class_destroy(myclass);
    unregister_chrdev(major, "mydevice");
}
```

模块的卸载函数与模块的加载函数是相对的操作，需要调用device_destory()和class_destory()对模块加载创建的myclass和mydevice进行销毁，然后调用unregister_chrdev()将动态分配的主设备号进行释放。

驱动程序完整代码如下：

程序代码[选择] Expand

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/sysfs.h>
#include <linux/string.h>
#include <linux/device.h>
#include <linux/fs.h>

static char mybuf[100] = "mydevice";
static int major;
static struct class *myclass;

static ssize_t mydevice_show(struct device *dev, struct device_attribute *attr,
        char *buf)
```

4、测试结果

将驱动程序进行编译后，将驱动模块进行加载，并查看创建出来的属性文件，使用cat和echo命令进行读写测试：

```
# make
# insmod simple-device.ko
# cd /sys/devices/virtual/myclass/mydevice/
# ls -al ./
```

可以看到创建出来的属性文件mydevice：

使用cat和echo命令进行文件读写测试：

```
# cat mydevice
# echo "I am a simplest driver." > mydevice
# cat mydevice
```

可以看到，能使用cat和echo命令正常完成属性文件的读写了：

5、小节

本文简单介绍了Linux内核中DEVICE_ATTR宏的实现，并使用一个简单的驱动程序示例来介绍了如何在Linux驱动程序中使用DEVICE_ATTR宏以及实现属性文件的读写函数。