sysfs The filesystem for exporting kernel objects
Documentation/filesystems/sysfs.txt

Patrick Mochel <mochel@osdl.org>
Mike Murphy <mamurph@cs.clemson.edu>
Revised: 16 August 2011
Original: 10 January 2003

Translator: Bill Wang(Liang) (bill.liangwlw@gmail.com,https://github.com/billwangwl)

What it is:
sysfs是一个ram-based filesystem，基于ramfs。用于导出kernel的数据结构，属性以及kernel space和user space之间的联系。

sysfs和kobject紧密相关。请查阅Documentation/kobject.txt来了解kobject的信息。

Using sysfs
sysfs在CONFIG_SYSFS选项打开后，编译到内核。通过mount来挂载：

```
mount -t sysfs sysfs /sys
```

Directory Creation
每一个注册到系统中的kobject都会有一个sysfs的目录。这个目录是对应Koject的父目录的子目录。【译注：这句话有点绕，有点递归的意思，意思是说，每个kobject都有一个目录，如果kobject有一个parent，那么他的目录就是parent对应的目录的子目录，如果么没有parent，那么就是/sys的子目录】

通过目录结构来标示kobject的层次关系。

Sysfs内部持有一个指针，该指针指向一个在sysfs_dirent对象关联的目录中实现目录的kobject【？】。在过去，Kobject pointer被sysfs用来在kobject 打开或者关闭的时候做引用计数。在现在的sysfs实现中，kobject的引用计数仅仅通过sysfs_schedule_callback()来修改。

Attributes
kobject的属性以文件的形式导出到文件系统中。sysfs将文件I/O操作转换为kobject属性中定义的方法。

属性应该是ASCII文件，建议每个文件只有一个值。不过每个文件只有一个值，效率不是很好，所以也可以有一组相同类型的值。

混合类型，多组数据，或者很复杂的格式化数据不是很恰当。

一个属性对的定义很简单：

```
struct attribute {
        char * name;
        struct module *owner;
        umode_t mode;
        };

int sysfs_create_file(struct kobject * kobj, const struct attribute * attr);
void sysfs_remove_file(struct kobject * kobj, const struct attribute * attr);
```

单单一个属性不能够对属性进行读或者写。各模块需要定义自己的attribute结构，添加对应的读写方法。

比如设备模型使用的strcut device_attribute：

```
struct device_attribute {
        struct attribute attr;
        ssize_t (*show)(struct device *dev, struct device_attribute *arr, char *buf);
        ssize_t (*store)(struct device *dev, struct device_attribute *attr, const char *buf,
size_t count);
```

同时也定义了宏来简化定义：

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

这样声明一个属性就很简单：

```
static DEVICE_ATTR(foo, S_IWUSR | S_IRUGO, show_foo, store_foo);这句话等同于：
static struct device_attribute dev_attr_foo = {
        .attr = {
                .name = "foo",
                .mode = S_IWUSR | S_IRUGO,
        },
        .show = show_foo,
        .store = store_foo,
};
```

Subsystem-Specific Callbacks

当一个模块定义一个新的属性类型，那么就必须实现对应的sysfs操作，这些操作用来读写属性文件。

```
struct sysfs_ops {
        ssize_t (*show)(struct kobject*, struct attribute *, char*);
        ssize_t (*store)(struct kobject*, struct attribute *, const char *, size_t);
}
```

[子系统应该已经定义了struct kobj_type，这kobj_type中包含了sysfs_ops，参考kobject的文档来查看更多细节]

当一个文件被读写，sysfs就会调用合适的方法。该方法会向通用的struct kobject 和 struct attribute pointers转换为特性的指针类型，然后调用关联的方法。

例子：
drivers/base/core.c

```
#define to_dev(obj) container_of(obj, struct device, kobj)
【译注：返回obj所在的struct device的指针】
#define to_dev_attr(_attr) container_of(_attr, struct device_attribute, attr)
【译注：返回attr所在的struct device_attribute的指针】

static ssize_t dev_attr_show(struct kobject *kobj, struct attribute* attr, char *buf)
{
        struct device_attribute *dev_attr = to_dev_attr(attr);
        struct device *dev = to_dev(kobj);
        ssize_t ret = -EIO;

        if (dev_attr->show)
                ret = dev_attr->show(dev, dev_attr, buf);
        if (ret >= (ssize_t)PAGE_SIZE) {
                print_symbol("dev_attr_show: %s return bad count\n", (unsigned
long)dev_attr->show);
        }
        return ret;
}
```

【？我的例子里，没有注册设备，只是注册是kobject，为何也能找到对应的show, store，是不是kobject只能用于设备模型，而不能用于其他方面】

Reading/Writing Attribute Data
为了读写属性，show()和store()方法必须在声明属性的时候定义。

```
ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t (*store)(struct device *dev, struct device_attribute *att, const char *buf, size_t
count);
```

换句话说，这两个方法参数只能有一个对象，一个属性和一个buf。

sysfs申请PAGE_SIZE大小的buffer，然后把这个buf传递给方法。每次读写，sysfs只会调用一次方法。这样在方法实现上就有这些特点：

---对于read(2)，show应该填满整个buffer。由于一个属性只能export一个值，或者一组同类型的值，所以buffer不会太大。

这个办法也可以让用户空间进行部分读和seek操作。如果用户空间通过seek到0，或者通过pread(2)时用参数0，那么show()多调用一次。

---对于write(2)，sysfs会将写buffer传入到store()。

当写sysfs文件时，用户空间应该先读取文件的值，修改成期望的值，然后把整个buf写回。

属性的读写操作应该操作同一个buf。

【？？？】为何写入writehere时，store()方法调用了两次？ echo writehere>/sys/hwobj/write_node

其他需要注意的：

---【？？？】写操作会引起show()的位置回到文件开始。

---buffer的大小总是PAGE_SIZE。

---show()应该返回写入到buf中的长度，返回值和scnprintf()一样。

【译注】

scnprintf -- Format a string and place it in a buffer

int scnprintf(char* buf, size_t size, const char *fmt, ...);

buf: the buffer to place the result into

size: the buffer size, including the trailing null space

【end】

---store()应该返回buffer所用的大小。如果所有的buffer都用完了，只需要返回count参数？

---show()或者store()方法可以返回错误。如果一个错误的参数传入，那么可以返回一个错误。

---show()或者store()传入的对象，会在sysfs引用的object的内存中（？）。所以，物理的设备可能不会存在，所以，如果需要，要检查物理设备是否存在。

简单的实现可以这样：

```
static ssize_t show_name(struct device *dev, struct device_attribute *attr, char *buf)
{
        return scnprintf(buf, PAGE_SIZE, "%s\n", dev->name);
}

static ssize_t store_name(struct device *dev, struct device_attribute *attr, const char *buf, size_t count)
{
        snprintf(dev->name, sizeof(dev->name, "%.*s", (int)min(count, sizeof(dev->name) - 1), buf);
        return count;
}

static DEVICE_ATTR(name, S_IRUGO, show_name, store_name);
```

(注意：真实的应用中，不应该通过userspace来设置device的名字。

Top Level Directory Layout

sysfs的目录结构导出了kernel的数据结构关系。
顶层的sysfs目录结构类似于：

block/
bus/
class/
dev/
devices/
firmware/
net/
fs/

devices/用一个文件系统标示了设备树。直接映射了内部的内核设备树，也就是struct device的结构。

bus/用罗列了内核中各种bus类型。每一种bus的目录中的都有两个文件夹：

devices/
drivers/

/sys/bus/[]/devices/ 包含了在系统中发现的设备的链接，指向了/sys/devices下的设备。
/sys/bus/[]/drivers/ 包含了bus上挂在的设备的驱动。

fs/包含了当前系统的filesystem，子目录时名子文件系统想要导出的属性。

dev/包含了两个文件夹，char/和block/。在各自的目录中，分别时用<major>:<minor>命名的符号链接。

更多关于设备模型的特性可以在/Documentation/driver-model/查看。

TODO: Finish this section

Current Interfaces
- devices (include/linux/device.h)

Structure:

```
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);
};
```

Declaring:

DEVICE_ATTR(_name, _mode, _show, _store);

Creation/Removal:

int device_create_file(struct device *dev, const struct device_attribute * a ttr);
void device_remove_file(struct device *dev, const struct device_attribute * attr);

- bus drivers (include/linux/device.h)

Structure:

struct bus_attribute {
struct attribute attr;
ssize_t (*show)(struct bus_type *, char * buf);
ssize_t (*store)(struct bus_type *, const char * buf, size_t count);
};

Declaring:

BUS_ATTR(_name, _mode, _show, _store)

Creation/Removal:

int bus_create_file(struct bus_type *, struct bus_attribute *);
void bus_remove_file(struct bus_type *, struct bus_attribute *);

- device drivers (include/linux/device.h)

Structure:

struct driver_attribute {
struct attribute attr;
struct attribute attr;
ssize_t (*show)(struct device_driver *, char * buf);
ssize_t (*store)(struct device_driver *, const char * buf,
size_t count);
};

Declaring:

DRIVER_ATTR(_name, _mode, _show, _store)

Creation/Removal:

int driver_create_file(struct device_driver *, const struct driver_attribute *);
void driver_remove_file(struct device_driver *, const struct driver_attribut e *);

Documentation
~~~~~~~~~~~~~
sysfs目录结构和属性定义了kernel和userspace之间的ABI。因此上，这些ABI要稳定和恰当的说明。所有新的sysfs attribute要在Documentation/ABI中说明，查看
Docomentation/ABI/README来了解更多。

【内核中用字符设备，block设备等，用于内核管理，通过udev和sysfs导出设备到用户空间】
~~~~~~~~~~~~~