## sysfs文件系统结构

sysfs是一个基于内存的文件系统，它的作用是将内核信息以文件的方式提供给用户程序使用。该文件系统的目录层次结构严格按照内核的数据结构组织。除了二进制文件外（只有特殊场合才使用），sysfs文件内容均以ASCII格式保存，且一个文件只保存一个数据，另外，一个文件不可大于一个内存页。

sysfs\_dirent是组成sysfs单元的基本数据结构，它是sysfs文件夹或文件在内存中的代表。sysfs\_dirent只表示文件类型（文件夹/普通文件/二进制文件/链接文件）及层级关系，其它信息都保存在对应的inode中。我们创建或删除一个sysfs文件或文件夹事实上只是对以sysfs\_dirent为节点的树的节点的添加或删除。sysfs\_dirent数据结构如下：

struct sysfs\_dirent {

       atomic\_t                s\_count;

       atomic\_t                s\_active;

       struct sysfs\_dirent  \*s\_parent;  /\* 指向父节点 \*/

struct sysfs\_dirent  \*s\_sibling;  /\* 指向兄弟节点 \*/

       const char             \*s\_name;    /\* 节点名称 \*/

       union {

              struct sysfs\_elem\_dir            s\_dir;      /\* 文件夹 \*/

              struct sysfs\_elem\_symlink    s\_symlink;  /\* 链接 \*/

              struct sysfs\_elem\_attr           s\_attr;     /\* 普通文件 \*/

              struct sysfs\_elem\_bin\_attr     s\_bin\_attr;  /\* 二进制文件 \*/

       };

       unsigned int           s\_flags;

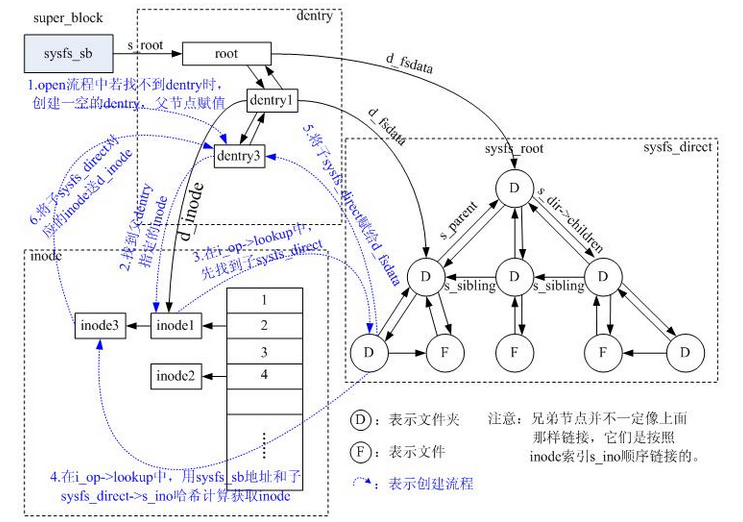
ino\_t          s\_ino;

       umode\_t                s\_mode;

       struct iattr             \*s\_iattr;

};

sysfs文件系统结构：



sysfs超级块sysfs\_sb、dentry根目录root、sysfs\_dirent根目录sysfs\_root都是在sysfs初始化时创建。

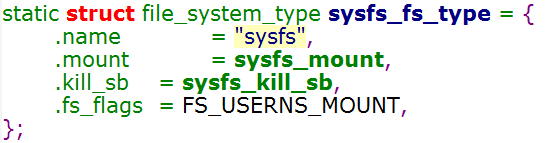
sysfs\_root下的子节点是添加设备对象或对象属性时调用sysfs\_create\_dir/ sysfs\_create\_file创建的，同时会申请对应的inode的索引号s\_ino。注意此时并未创建inode。

inode是在用到的时候调用sysfs\_get\_inode函数创建并依据sysfs\_sb地址和申请到的s\_ino索引计算散列表位置放入其中。

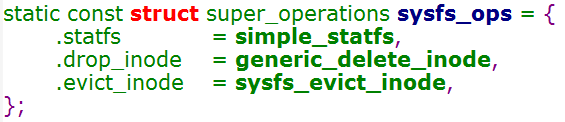
dentry的子节点也是需要用的时候才会创建。比如open文件时，会调用path\_walk根据路径一层层的查找指定dentry，如果找不到，则创建一个，并调用父dentry的inode的lookup函数（sysfs文件系统的为sysfs\_lookup）查找对应的子inode填充指定的dentry。

## VFS提供给sysfs文件系统的接口

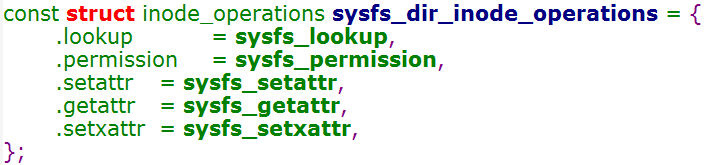
1. 文件系统类型：为了将sysfs文件系统成功挂载，必须在内核中具有file\_system\_type结构体，该结构指出文件系统的名称、挂载方法等。



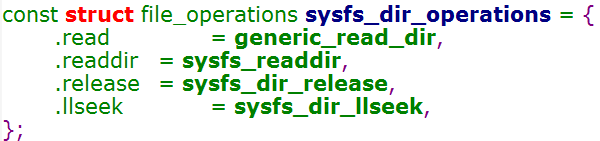
1. 超级块操作：所含操作有查看文件系统信息、删除inode、回收inode。



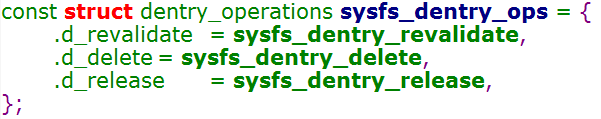
1. inode操作：所含操作包括查找inode、检查inode权限、设置inode属性、获取inode属性、设置inode扩展属性



1. 文件操作：所含操作有读文件、释放文件、获取文件偏移量，sysfs文件系统中具体的文件内容存放在open系统调用之后创建的file结构体中。



1. 目录项操作：所含操作有验证目录项(是否被删除、修改、重命名等)、删除目录项、释放目录项



## 数据对象kobject

Kobject 是Linux 2.6引入的新的设备管理机制。通过这个数据结构使所有设备在底层都具有统一的接口，kobject提供基本的对象管理，是构成Linux2.6设备模型的核心结构，它与sysfs文件系统紧密关联，每个在内核中注册的kobject对象都对应于sysfs文件系统中的一个目录。Kobject是组成设备模型的基本结构。

Kobject结构定义为：  
struct kobject {  
 char \* k\_name; //指向设备名称的指针  
 char name[KOBJ\_NAME\_LEN]; //设备名称  
 struct kref kref; //对象引用计数  
 struct list\_head\_entry; //挂接到所在kset中去的单元  
 struct kobject \* parent; //指向父对象的指针  
 struct kset \* kset; //所属kset的指针  
 struct kobj\_type \* ktype; //指向其对象类型描述符的指针  
 struct dentry \* dentry; //sysfs文件系统中与该对象对应的文件节点路径指针

};

其中的kref域表示该对象引用的计数，内核通过kref实现对象引用计数管理，内核提供两个函数kobject\_get()、kobject\_put() 分别用于增加和减少引用计数，当引用计数为0时，所有该对象使用的资源释放。Ktype 域是一个指向kobj type结构的指针，表示该对象的类型。

* 相关函数：

void kobject\_init(struct kobject \* kobj)；kobject初始化函数。

int kobject\_set\_name(struct kobject \*kobj, const char \*format, ...)；//设置指定kobject的名称。

struct kobject \*kobject\_get(struct kobject \*kobj)；将kobj 对象的引用计数加1，同时返回该对象的指针。

void kobject\_put(struct kobject \* kobj)； 将kobj对象的引用计数减1，如果引用计数降为0，则调用kobject release()释放该kobject对象。

int kobject\_add(struct kobject \* kobj)；将kobj对象加入Linux设备层次。挂接该kobject对象到kset的list链中，增加父目录各级kobject的引用计数，在其 parent指向的目录下创建文件节点，并启动该类型内核对象的hotplug函数。

int kobject\_register(struct kobject \* kobj)；kobject注册函数。通过调用kobject init()初始化kobj，再调用kobject\_add()完成该内核对象的注册。

void kobject\_del(struct kobject \* kobj)；从Linux设备层次中删除kobj对象。

void kobject\_unregister(struct kobject \* kobj)；kobject注销函数。与kobject register()相反，它首先调用kobject del从设备层次中删除该对象，再调用kobject put()减少该对象的引用计数，如果引用计数降为0，则释放kobject对象。

* Kobj\_type

struct kobj\_type {

void (\*release)(struct kobject \*);

struct sysfs\_ops \* sysfs\_ops;

struct attribute \*\* default\_attrs;

};

Kobj\_type数据结构包含三个域：一个release方法用于释放kobject占用的资源；一个sysfs\_ops指针指向sysfs操作表和一个sysfs文件系统缺省属性列表。Sysfs操作表包括两个函数store()和show()。当用户态读取属性 时，show()函数被调用，该函数编码指定属性值存入buffer中返回给用户态；而store()函数用于存储用户态传入的属性值。

* attribute

struct attribute {

char \* name;

struct module \* owner;

mode\_t mode;  
};

attribute, 属性。它以文件的形式输出到sysfs的目录当中。在kobject对应的目录下面。文件名就是name。文件读写的方法对应于kobj\_type中的sysfs\_ops。

* kset

kset最重要的是建立上层(sub-system)和下层的 (kobject)的关联性。kobject 也会利用它了分辨自已是属于那一个类型，然后在/sys 下建立正确的目录位置。而kset 的优先权比较高，kobject会利用自已的\*kset 找到自已所属的kset，並把\*ktype 指定成该kset下的ktype，除非没有定义kset，才会用ktype来建立关系。Kobject通过kset组织成层次化的结构，kset是具有相 同类型的kobject的集合，在内核中用kset数据结构表示。

## sys文件系统读写

每当系统增加一个设备时，在sys文件系统中的表现就是新增一个kobject结构，同时会在/sys下创建一个目录。该过程的流程是：

kobject\_add()->create\_dir()->sysfs\_create\_dir()。

sysfs就是利用VFS的接口去读写kobject的层次结构，建立起来的文件系统。 kobject的层次结构的注册与注销XX\_register()形成的。文件系统是个很模糊广泛的概念， linux把所有的资源都看成是文件，让用户通过一个统一的文件系统操作界面，也就是同一组系统调用，对属于不同文件系统的文件进行操作。这样，就可以对用户程序隐藏各种不同文件系统的实现细节，为用户程序提供了一个统一的，抽象的，虚拟的文件系统界面，这个抽象出来的接口就是一组函数操作。

实现一种文件系统就是要实现VFS所定义的一系列接口，file\_operations, dentry\_operations, inode\_operations等，供上层调用。file\_operations是描述对每个具体文件的操作方法(如：读，写)，dentry\_operations结构体指明了VFS所有目录的操作方法, 而inode\_operations提供所有结点的操作方法。

举个例子，我们写C程序，open(“hello.c”, O\_RDONLY)，它通过系统调用的流程是这样的：

open() -> 系统调用->sys\_open() -> filp\_open()-> dentry\_open() -> file\_operations->open()；

不同的文件系统，调用不同的file\_operations->open()，在sysfs下就是sysfs\_open\_file()。

sysfs是典型的特殊文件。它存储的信息都是由系统动态的生成的，它动态的包含了整个机器的硬件资源情况。从sysfs读写就相当于向 kobject层次结构提取数据。

sysfs文件系统读文件的流程：

sysfs\_read\_file:

1. struct sysfs\_buffer \* buffer = file->private\_data;
2. 调用sys\_ops->show(kobj, attr\_sd->s\_attr.attr, buffer->page);
3. 调用simple\_read\_from\_buffer将读取到的内容拷贝到用户空间(copy\_to\_user)。

第三步中的sys\_ops是通过kobj->ktype->sysfs\_ops获得的

## 数据对象kobject的变化

1. kobject的创建

在系统增加一个设备时，调用device\_add（），该函数调用kobject\_add（）创建一个数据对象kobject。通过数据对象kobject，可以通过sys\_create\_dir()在sysfs文件系统下创建该设备的相应目录结构sysfs\_dirent。

1. kobject的删除