linux module

一、概念

1.内核中通过模块机制让用户动态扩展系统功能,减小linux内核体积，节省flash。

2.使用MODULE\_LICENSE（）;声明模块的发行许可类型，常用的有GPV, GPL v2, Dual BSD/GPL等。

3.内核模块编程中没有浮点数预算功能，也不具备浮点数输出功能。

4.内核模块不能使用标准C中提供的任何库函数，使用的函数都是内核源码中实现的。

5.linux内核模块可以添加内核驱动菜单，通过make menuconfig进入配置为m，重新make modules编译得到.ko文件。

二、如何编写一个模块

1.借助/lib/modules/`uname -r`/build中的Makefile文件编译内核模块

2.使用make modules编译内核源码树中的模块。

模块编译要依赖具体的源代码及配置，要得到模块，所使用的源码必须要编译完成才可以执行make modules命令

驱动操作命令：

insmod / modprobe 加载驱动

rmmod 卸载驱动

lsmod 查看系统中所有已经被加载了的所有的模块以及模块间的依赖关系

modinfo 获得模块的信息

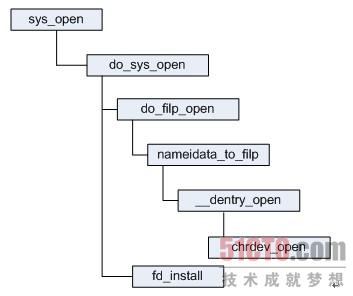
查看已经加载的驱动模块的信息：

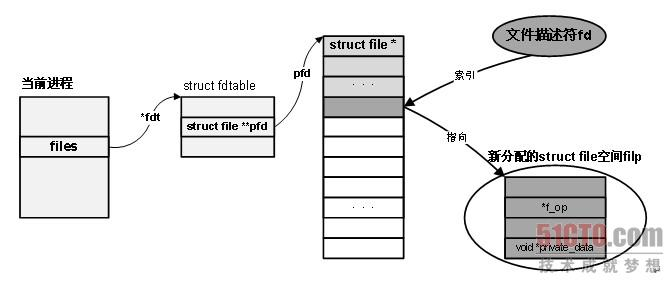
lsmod 能够显示驱动的大小以及被谁使用

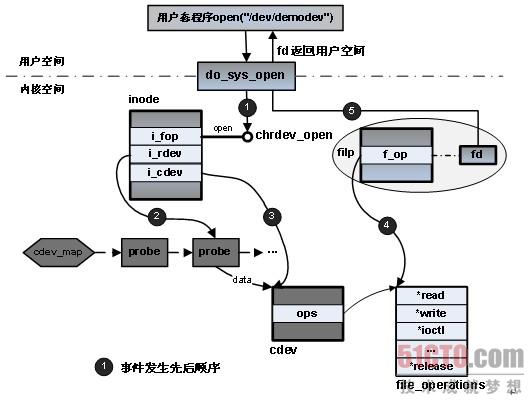
cat /proc/modules 能够显示驱动模块大小、在内核空间中的地址

cat /proc/devices 只显示驱动的主设备号，且是分类显示

/sys/modules 下面存在对应的驱动的目录，目录下包含驱动的分段信息等等。





Linux设备模型的核心思想是（通过xxx手段，实现xxx目的）：

1. 用Device（struct device）和Device Driver（struct device\_driver）两个数据结构，分别从“有什么用”和“怎么用”两个角度描述硬件设备。这样就统一了编写设备驱动的格式，使驱动开发从论述题变为填空体，从而简化了设备驱动的开发。

2. 同样使用Device和Device Driver两个数据结构，实现硬件设备的即插即用（热拔插）。

在Linux内核中，只要任何Device和Device Driver具有相同的名字，内核就会执行Device Driver结构中的初始化函数（probe），该函数会初始化设备，使其为可用状态。

而对大多数热拔插设备而言，它们的Device Driver一直存在内核中。当设备没有插入时，其Device结构不存在，因而其Driver也就不执行初始化操作。当设备插入时，内核会创建一个Device结构（名称和Driver相同），此时就会触发Driver的执行。这就是即插即用的概念。

3. 通过"Bus-->Device”类型的树状结构解决设备之间的依赖，启动某一个设备前，内核会检查该设备是否依赖其它设备或者总线，如果依赖，则检查所依赖的对象是否已经启动，如果没有，则会先启动它们，直到启动该设备的条件具备为止。而驱动开发人员需要做的，就是在编写设备驱动时，告知内核该设备的依赖关系即可。

4. 使用Class结构，在设备模型中引入面向对象的概念，这样可以最大限度地抽象共性，减少驱动开发过程中的重复劳动，降低工作量。

<https://blog.csdn.net/qq_16777851/category_7901554.html>

<http://www.wowotech.net/>