**驱动移植第一天**

**一、常用命令：**

加载模块：sudo insmod xxx.ko

查看系统中模块：lsmod |grep xxx

卸载模块：sudo rmmod xxx

查看加载模块和卸载模时printk打印的信息：dmesg 或dmesg |tail

查看设备号：cat /proc/devices

创建设备文件：sudo mknod /dev/设备文件名 c或者b 主设备号 次设备号

**二、模块化编程：**

Linux的开发者，遍布世界各地，大牛们设计Linux内核的时候，融入了模块化的思想，就是说，现在大家有一个现成的Linux操作系统，所有的开发者写的代码对于这个Linux操作系统而言都是一个模块，开发者可以以模块的形式将自己的代码添加到内核，也可以从操作系统中卸载自己的模块。

**三、模块化编程步骤：**



**四、Linux 内核打印函数printk：**

printk的用法和printf类似，printf用于用户空间，printk用于内核空间。

可以通过cat /proc/sys/kernel/printk这个文件，查看系统默认的日志级别。

**五、模块的编译：**



**六、Linux 驱动与裸机驱动的区别：**

**1.裸机驱动**

直接操作控制器，让硬件工作，实现硬件功能和如何使用这些功能是在一起

**2.Linux 驱动**

因为操作系统中，同类硬件设备一般只有一个，但是想操作硬件设备的进程会很多。这就带来一个问题，应用层如何访问硬件设备以及多个人同时访问的时候，如何解决并发问题。

Linux 驱动本质是向应用层提供访问硬件设备的函数接口，也就是说驱动只是提供硬件的功能函数接口，而如何使用这些功能由应用层代码去做。

注意:Linux下的驱动提供函数接口，必须遵从设备驱动的框架,便于Linux操作系统对设备进行管理。

Linux 驱动 = Linux 驱动框架 + 裸机操作

**七、Linux 设备分类：**

**问：什么是设备文件，设备文件的作用，设备节点、设备驱动及设备的关联？**



应用程序使用设备实现过程

* 1. 操作系统与外部设备(打印机和终端等)都是通过设备文件来进行通信的，在Linux系统与外部设备通讯之前，这个设备必须首先要有一个设备文件，设备文件均放在/dev目录下。一般情况下在安装系统的时候系统自动创建了很多已检测到的设备文件，但有时候我们也需要手动创建，命令行生成设备文件的方式有insf、mksf、mknod等。
  2. 字符设备通过insmod后，只是单纯的加载了驱动，可以通过cat对/proc/devices进行查看，/proc不是一个真实存在的文件夹，而是对内存情况的一个反映，提供一个接口而已。这个cat出的信息中，前面的数字是注册设备时获得的主设备。如果想获得对设备的引用，需使用mknod命令在文件系统上创建一个设备节点，如命令：mknod /dev/led\_device c 250 0创建一个字符设备(c),主设备号是250，次设备号是0。（一旦用mknod生成了一个特别的设备文件，它就永远存在了硬盘上，触发自己通过“rm”命令来删除它）
  3. 当访问一个设备节点时，系统是如何知道使用哪个设备驱动及访问哪个设备？这个是通过设备号来实现，当我们创建一个设备节点时需要指定主设备号和次设备号。对于设备节点来说，名字不是重要的，设备号才是最重要的，它实际指定了对应的驱动程序和对应的设备。
  4. Linux的设备管理是和文件系统紧密结合的，各种设备都以文件的形式存放在/dev目录下，称为设备文件。应用程序可以打开、关闭和读写这些设备文件，系统为设备编了号，每个设备号分为主设备号和次设备号。主设备号用来区分不同种类的设备，而次设备号用来区分同一类型的多个设备。
  5. Linux为所有的设备文件都提供了统一的操作函数接口，方法是使用数据结构struct file\_operations。这个数据结构中包括许多操作函数的指针，如open()、close()、read()、和write()等，但由于外设的种类较多，操作的方式各不相同。对于各种设备文件，则最终调用各自驱动程序中的I/O函数进行具体设备的操作。所以应用程序根本不用考虑操作的是设备还是普通文件，可一律当文件处理，具有非常清晰统一的I/O接口。所以file\_operations是文件层次的I/O接口。
  6. 设备号记录在设备文件中

设备号(32bit) = 主设备号(12bit) + 次设备号(20bit)

主设备号 : 代表这一类设备的驱动程序

次设备号 : 为了区分同类型设备的不同设备

(3个u盘设备，都是u盘设备，但是通过次设备号区分不同的u盘)

总结：设备文件为操作系统与外部设备通讯的接口，设备驱动为该设备文件与操纵系统通讯的具体实现。操作系统根据唯一的32位设备号（高12位为主设备号+低20位为次设备号）确定操作的是哪类设备中哪个成员。

**1.字符设备：以字节为单位读写的设备**

键盘，鼠标, ...

通过字符设备驱动程序可以实现字符(char)设备像字节流(类似文件)一样被访问，字符设备驱动程序通常至少实现open、close、read和write的系统调用。

**2.块设备：以块为单位（效率最高）读写的设备**

u盘，emmc,硬盘,....

块设备(如磁盘)上能够容纳filesystem，块设备和字符设备的区别仅仅在于内核管理内部数据的方式，也就是内核及驱动程序之间的软件接口。

**3.网络设备：用于网络通讯的设备**

网卡

内核和网络设备驱动程序间的通信，完全不同于内核和字符及块驱动程序之间的通信，内核调用一套和数据包相关的函数而不是read、write等。

注意:

(1)字符设备和块设备驱动在应用存在设备文件和驱动对应。

(2)网络设备驱动在应用层存在是网络接口 (ifconfig命令可以查看)

**八、应用层访问驱动程序：**

Linux的世界里一切皆文件，所有的硬件设备操作到应用层都会被抽象成文件的操作，如果应用层要访问硬件设备，必定要调用到硬件对应的驱动程序。



在Linux系统里：

1. 每个文件都用一个struct inode结构体来记录这个文件的所有信息，如文件类型和访问权限等。
2. 每个驱动程序在应用层的/dev目录下都会有一个设备文件和它对应。
3. 每打开一次文件，在VFS层都会分配一个struct file结构体来描述打开的这个文件。

注意：struct inode描述的是文件的静态信息，即这些信息很少被改变；而struct file结构体描述的是动态信息，如在对文件操作的时候，strcut file里面的信息经常会发生变化，如f\_pos记录的是当前文件的位偏移量。

struct inode : 描述设备文件的属性信息，只需要分配一次 (记录的静态信息)

struct file : 当打开设备文件的时候，就会分配一次 (记录动态信息,例如:文件偏移量)

注意：

struct file结构体对应就是我们的文件描述符

想访问底层设备，就必须打开对应的设备文件，就是在这个打开的过程中，Linux内核将应用层和对应的驱动程序关联起来。

1. 当open函数打开设备文件时，可以根据设备文件对应的struct inode结构体描述的信息，可以知道接下来要操作的设备类型（字符设备还是块设备），还会分配一个struct file结构体。
2. 根据struct inode结构体里面记录的设备号，可以找到对应的驱动程序。这里以字符设备为例。在Linux操作系统中每个字符设备有一个strcut cdev结构体。此结构体描述了字符设备的所有信息，其中最重要一项就是字符设备的操作函数接口。
3. 找到struct cdev结构体后，Linux内核就会将strcut cdev结构体所在的内存空间首地址记录在struct inode结构体的i\_cdev成员中。将strcut cdev结构体中记录的函数操作接口地址记录在strcut file结构体的f\_op成员中。
4. 任务完成，VFS层会给应用层返回一个文件描述符（fd），这个fd是和struct file结构体对应的。接下来上层的应用程序就可以通过fd来找到struct fie，然后由struct file找到操作字符设备的函数接口。

**九、字符设备驱动程序：**

**1.针对自己的字符设备设计结构体**

struct xxx\_device

{

//通用的字符设备的信息

**struct cdev** cdev;

//自己字符设备的信息

.....

};

**2.实现设备的函数接口,记录在struct file\_operations**

如：xxx\_open xxx\_read xxx\_write xxx\_release

static const struct **file\_operations** xxx\_registers\_fops = {

.open = xxx\_open,

.read = xxx\_read,

.write = xxx\_write,

.release = xxx\_release,

.owner = THIS\_MODULE,

};

**3.模块的入口函数**

[1]分配自己字符设备结构体空间 (**kmalloc**)

[2]初始化通用字符设备，让cdev结构体记录struct file\_operations (**cdev\_init**)

[3]注册设备号(**register\_chardev\_region**)

[4]添加字符设备到操作系统(**cdev\_add**)

**4.模块的出口函数**

[1]删除字符设备(**unregister\_chrdev\_region**)

[2]注销设备号(**cdev\_del**)

[3]释放空间(**kfree**)