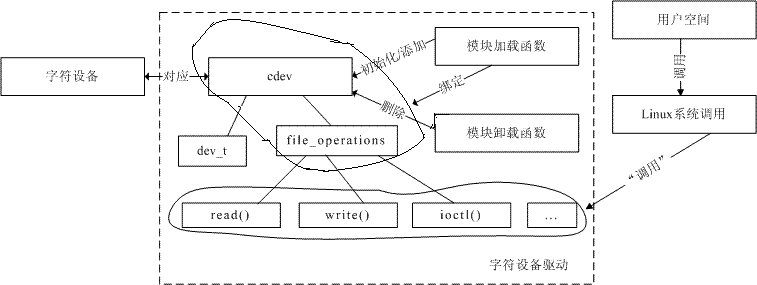
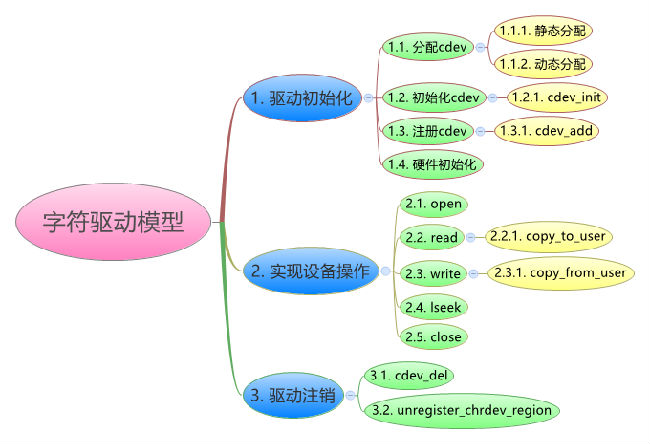
# Linux驱动\_LED

## 1.字符设备框架图

## 2.LED驱动基本代码

### 1.1基本头文件

#include <linux/module.h> //最基本的文件，支持动态添加和卸载模块。Hello World驱动要这一个文件就可以了

#include <linux/fs.h> //包含了文件操作相关struct的定义，例如大名鼎鼎的struct file\_operations

                      //包含了struct inode 的定义，MINOR、MAJOR的头文件。

#include <linux/kernel.h> //printk()

#include <linux/slab.h>　 //kmalloc()

#include <asm/uaccess.h>　//copy\_to\_user和copy\_from\_user

#include <linux/device.h> //包含了device、class等结构的定义,class\_create

#include <linux/io.h>     //包含了ioremap、iowrite等内核访问IO内存等函数的定义。

#include <linux/cdev.h> //cdev\_init等函数定义

#include <asm/arch/map.h> //包含了S3C24XX\_VA\_GPIO定义

### 1.2 驱动入口函数

注册init和exit函数

module\_init**(**leds\_drv\_init**);**

module\_exit**(**leds\_drv\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

其中module\_init的定义如下：

#define module\_init(x) \_\_initcall(x);

#define \_\_initcall(fn) device\_initcall(fn)

#define device\_initcall(fn) \_\_define\_initcall("6",fn,6)

在linux系统启动的时候，会依次调用这些注册函数。

### 1.3 全局变量

#define DEVICE\_NAME "leds"

#define MAX\_LEDS\_NUM 3

//leds结构体

//status: 1 亮 0 灭

struct leds\_dev\_t

**{**

struct cdev cdev**;**

int status**[**MAX\_LEDS\_NUM**];**

**}**leds\_dev**;**

//led1/2/3结构体

//status: 1 亮 0 灭

struct led\_dev\_t

**{**

struct cdev cdev**;**

int status**;**

**}** **\***led\_dev**;**

//class结构体，用于自动生成设备节点

static struct class **\***ledsdrv\_class**;**

//gpio寄存器虚拟地址

volatile unsigned long **\***gpfcon **=** **NULL;**

volatile unsigned long **\***gpfdat **=** **NULL;**

//驱动主设备号

static int leds\_major**;**

//led字符设备驱动对应的操作函数

static struct file\_operations leds\_drv\_fops **=** **{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** leds\_drv\_open**,**

**.**write **=** leds\_drv\_write**,**

**.**read **=** leds\_drv\_read**,**

**.**ioctl **=** leds\_drv\_ioctl**,**

**};**

### 1.4 leds\_drv\_init

这里一共创建了leds,led1/2/3四个设备，leds用于同时操作3个led灯，led1/2/3分别操作对应的led灯。  
初始化主要包含驱动设备号申请-->字符设备注册-->class创建-->获取gpio寄存器虚拟地址四个步骤。

static int leds\_drv\_init**(**void**)**

**{**

int i**;**

dev\_t dev\_id**;**

//1.申请设备号

//注册字符设备编号，在proc/devices中创建leds选项

**if(**leds\_major**)**

**{**

//静态

dev\_id **=** MKDEV**(**leds\_major**,**0**);**

register\_chrdev\_region**(**dev\_id**,(**1**+**MAX\_LEDS\_NUM**),**DEVICE\_NAME**);**

**}**

**else**

**{**

//动态

alloc\_chrdev\_region**(&**dev\_id**,**0**,(**1**+**MAX\_LEDS\_NUM**),**DEVICE\_NAME**);**

leds\_major **=** MAJOR**(**dev\_id**);**

**}**

//2.注册字符设备

//初始化并且注册leds cdev

cdev\_init**(&**leds\_dev**.**cdev**,** **&**leds\_drv\_fops**);**

leds\_dev**.**cdev**.**owner **=** THIS\_MODULE**;**

cdev\_add**(&**leds\_dev**.**cdev**,**dev\_id**,**1**);**

//初始化并注册led0/1/2/3

led\_dev **=** kmalloc**(**MAX\_LEDS\_NUM**\*sizeof(**struct led\_dev\_t**),**GFP\_KERNEL**);**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

dev\_id **=** MKDEV**(**leds\_major**,**i**+**1**);**

cdev\_init**(&**led\_dev**[**i**].**cdev**,** **&**leds\_drv\_fops**);**

cdev\_add**(&**led\_dev**[**i**].**cdev**,** dev\_id**,**1**);**

**}**

//3.自动创建设备节点

//创建一个类，这个类存放于sysfs下面 /sys/class/leds

ledsdrv\_class **=** class\_create**(**THIS\_MODULE**,** DEVICE\_NAME**);**

//mdev daemon就会自动创建节点/dev/leds

//加载模块的时候，用户空间中的mdev会自动响应device\_create(…)函数，去/sysfs下寻找对应的类从而创建设备节点。

class\_device\_create**(**ledsdrv\_class**,** **NULL,** MKDEV**(**leds\_major**,** 0**),** **NULL,** DEVICE\_NAME**);**

//创建节点/dev/led1/2/3

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

class\_device\_create**(**ledsdrv\_class**,** **NULL,** MKDEV**(**leds\_major**,** i**+**1**),** **NULL,** "led%d"**,** i**);**

**}**

//4.虚拟地址动态映射

gpfcon **=** **(**volatile unsigned long **\*)**ioremap**(**0x56000050**,** 16**);**

gpfdat **=** gpfcon **+** 1**;**

//也可以采用静态的方法

//gpfcon = S3C24XX\_VA\_GPIO+0x50;

//gpfdat = gpfcon + 1;

**return** 0**;**

**}**

当insmod该驱动时，便会调用该函数创建对应的设备节点和文件。  
proc下创建的文件。

#cat /proc/devices

Character devices**:**

1 mem

2 pty

3 ttyp

4 **/**dev**/**vc**/**0

4 tty

4 ttyS

5 **/**dev**/**tty

5 **/**dev**/**console

5 **/**dev**/**ptmx

6 lp

7 vcs

10 misc

13 input

14 sound

29 fb

90 mtd

99 ppdev

116 alsa

128 ptm

136 pts

180 usb

189 usb\_device

204 s3c2410\_serial

252 leds

253 usb\_endpoint

254 rtc

Block devices**:**

1 ramdisk

7 loop

8 sd

31 mtdblock

sys下创建的文件

# ls -l /sys/class/

drwxr**-**xr**-**x 6 0 0 0 Jan 1 00**:**07 leds

# ls -l /sys/class/leds

drwxr**-**xr**-**x 2 0 0 0 Jan 1 00**:**07 led0

drwxr**-**xr**-**x 2 0 0 0 Jan 1 00**:**10 led1

drwxr**-**xr**-**x 2 0 0 0 Jan 1 00**:**07 led2

drwxr**-**xr**-**x 2 0 0 0 Jan 1 00**:**10 leds

#ls -l /sys/class/leds/led0

**-**r**--**r**--**r**--** 1 0 0 4096 Jan 1 00**:**07 dev

lrwxrwxrwx 1 0 0 0 Jan 1 00**:**20 subsystem **->** **../../../**class**/**leds

**--**w**-------** 1 0 0 4096 Jan 1 00**:**20 uevent

# cat /sys/class/leds/led0/dev

252**:**1

dev下的设备节点

# ls -l /dev/ | grep led

crw**-**rw**----** 1 0 0 252**,** 1 Jan 1 02**:**52 led0

crw**-**rw**----** 1 0 0 252**,** 2 Jan 1 02**:**52 led1

crw**-**rw**----** 1 0 0 252**,** 3 Jan 1 02**:**52 led2

crw**-**rw**----** 1 0 0 252**,** 0 Jan 1 02**:**52 leds

四个设备的主设备号为252，次设备号为0 1 2 3。

再注意一个地方，获取GPIOF寄存器的方式有两种，一种是ioremap动态映射方式，还有一种是静态映射方式，静态映射在系统初始化已经完成，对应的代码为：

static struct map\_desc s3c\_iodesc**[]** \_\_initdata **=** **{**

IODESC\_ENT**(**GPIO**),**

IODESC\_ENT**(**IRQ**),**

IODESC\_ENT**(**MEMCTRL**),**

IODESC\_ENT**(**UART**),**

IODESC\_ENT**(**IIS**)** // for IIS, thiswa.diy@163.com, www.100ask.net

**};**

### 1.5 leds\_drv\_exit

在rmmod时清理初始化创建的节点和文件。

static void leds\_drv\_exit**(**void**)**

**{**

int i**;**

//1.删除设备节点

//删除/dev/leds设备节点

class\_device\_destroy**(**ledsdrv\_class**,**MKDEV**(**leds\_major**,**0**));**

//删除/dev/led1/2/3设备节点

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

class\_device\_destroy**(**ledsdrv\_class**,**MKDEV**(**leds\_major**,**i**+**1**));**

**}**

class\_destroy**(**ledsdrv\_class**);**

//2.删除字符设备

//删除cdev

cdev\_del**(&**leds\_dev**.**cdev**);**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

cdev\_del**(&**led\_dev**[**i**].**cdev**);**

**}**

//3.释放设备号

unregister\_chrdev\_region**(**MKDEV**(**leds\_major**,**0**),**1**+**MAX\_LEDS\_NUM**);**

//4.释放虚拟io空间

iounmap**(**gpfcon**);**

kfree**(**led\_dev**);**

**}**

### 1.6 leds\_drv\_open

open以下设备/dev/leds,/dev/led0,/dev/led1,/dev/led2时，会调用下面函数。  
其中有两个地方需要注意：  
(1)从设备号，由于四个led设备公用一个open函数，可以通过次设备号来判断打开的是哪个设备并做对应的处理。  
(2)私有数据，四个led设备都拥有对应的私有数据，有自身的结构体，通过container\_of从cdev结构体获取其对应的包含cdev的结构体，里面包含了私有数据。

static int leds\_drv\_open**(**struct inode **\***inode**,** struct file **\***file**)**

**{**

struct leds\_dev\_t **\***dev1**;**

struct led\_dev\_t **\***dev2**;**

//获取从设备号

int minor **=** iminor**(**inode**);**

printk**(**"leds\_drv\_open,minor:%d\n"**,**minor**);**

**if(**minor **==** 0**)**

**{**

//用struct file的文件私有数据指针保存struct mycdev结构体指针

dev1 **=** container\_of**(**inode**->**i\_cdev**,**struct leds\_dev\_t**,**cdev**);**

file**->**private\_data **=** dev1**;**

//配置GPF4,5,6为输出

**\***gpfcon **&=** **~((**0x3**<<(**4**\***2**))** **|** **(**0x3**<<(**5**\***2**))** **|** **(**0x3**<<(**6**\***2**)));**

**\***gpfcon **|=** **((**0x1**<<(**4**\***2**))** **|** **(**0x1**<<(**5**\***2**))** **|** **(**0x1**<<(**6**\***2**)));**

**}**

**else**

**{**

//用struct file的文件私有数据指针保存struct mycdev结构体指针

dev2**=**container\_of**(**inode**->**i\_cdev**,**struct led\_dev\_t**,**cdev**);**

file**->**private\_data **=** dev2**;**

//否则单独设置某个引脚为输出功能

**\***gpfcon **&=** **~(**0x3 **<<** **((**minor**+**3**)\***2**));**

**\***gpfcon **|=** **(**0x1 **<<** **((**minor**+**3**)\***2**));**

**}**

**return** 0**;**

**}**

### 1.7 leds\_drv\_write

操作led亮灭，并更新对应的私有数据，保存led状态。

static ssize\_t leds\_drv\_write**(**struct file **\***file**,** const char \_\_user **\***buf**,** size\_t count**,** loff\_t **\*** ppos**)**

**{**

int i**;**

int val**;**

int minor **=** iminor**(**file**->**f\_path**.**dentry**->**d\_inode**);**

printk**(**"leds\_drv\_write,minor:%d\n"**,**minor**);**

copy\_from\_user**(&**val**,** buf**,** count**);**

**if(**minor **==** 0**)**

**{**

//获取私有数据

struct leds\_dev\_t **\***dev **=** file**->**private\_data**;**

**if** **(**val **==** 1**)**

**{**

//全部点亮

**\***gpfdat **&=** **~((**1**<<**4**)** **|** **(**1**<<**5**)** **|** **(**1**<<**6**));**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

dev**->**status**[**i**]** **=** 1**;**

**}**

**}**

**else**

**{**

//全部熄灭

**\***gpfdat **|=** **(**1**<<**4**)** **|** **(**1**<<**5**)** **|** **(**1**<<**6**);**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

dev**->**status**[**i**]** **=** 0**;**

**}**

**}**

**}**

**else**

**{**

struct led\_dev\_t **\***dev **=** file**->**private\_data**;**

**if(**val **==** 1**)**

**{**

//某个点亮

**\***gpfdat **&=** **~(**0x1**<<(**minor **+** 3**));**

dev**->**status **=** 1**;**

leds\_dev**.**status**[**minor**-**1**]** **=** 1**;**

**}**

**else**

**{**

//某个熄灭

**\***gpfdat **|=** **(**0x1**<<(**minor **+** 3**));**

dev**->**status **=** 0**;**

leds\_dev**.**status**[**minor**-**1**]** **=** 0**;**

**}**

**}**

**return** 0**;**

**}**

### 1.8 leds\_drv\_ioctl

通过ioctl操作led。

static int leds\_drv\_ioctl**(**struct inode **\***inode**,** struct file **\***file**,** unsigned int cmd**,**unsigned long arg**)**

**{**

int i**;**

int minor **=** iminor**(**inode**);**

printk**(**"leds\_drv\_ioctl,minor:%d\n"**,**minor**);**

**if(**minor **==** 0**)**

**{**

struct leds\_dev\_t **\***dev **=** file**->**private\_data**;**

**if** **(**cmd **==** 1**)**

**{**

//全部点亮

**\***gpfdat **&=** **~((**1**<<**4**)** **|** **(**1**<<**5**)** **|** **(**1**<<**6**));**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

dev**->**status**[**i**]** **=** 1**;**

**}**

**}**

**else**

**{**

//全部熄灭

**\***gpfdat **|=** **(**1**<<**4**)** **|** **(**1**<<**5**)** **|** **(**1**<<**6**);**

**for(**i **=** 0**;** i **<** MAX\_LEDS\_NUM**;** i**++)**

**{**

dev**->**status**[**i**]** **=** 0**;**

**}**

**}**

**}**

**else**

**{**

struct led\_dev\_t **\***dev **=** file**->**private\_data**;**

**if(**cmd **==** 1**)**

**{**

//某个点亮

**\***gpfdat **&=** **~(**0x1**<<(**minor **+** 3**));**

dev**->**status **=** 1**;**

leds\_dev**.**status**[**minor**-**1**]** **=** 1**;**

**}**

**else**

**{**

//某个熄灭

**\***gpfdat **|=** **(**0x1**<<(**minor **+** 3**));**

dev**->**status **=** 0**;**

leds\_dev**.**status**[**minor**-**1**]** **=** 0**;**

**}**

**}**

**}**

### 1.9 leds\_drv\_read

通过read读取led状态

ssize\_t leds\_drv\_read**(**struct file **\***file**,** char **\***buf**,** size\_t len**,** loff\_t **\***off**)**

**{**

int minor **=** iminor**(**file**->**f\_path**.**dentry**->**d\_inode**);**

printk**(**"leds\_drv\_read,minor:%d\n"**,**minor**);**

**if(**minor **==** 0**)**

**{**

struct leds\_dev\_t **\***dev **=** file**->**private\_data**;**

copy\_to\_user**(**buf**,** dev**->**status**,** MAX\_LEDS\_NUM**\*sizeof(**int**));**

**return** **(**MAX\_LEDS\_NUM**\*sizeof(**int**));**

**}**

**else**

**{**

struct led\_dev\_t **\***dev **=** file**->**private\_data**;**

copy\_to\_user**(**buf**,** **&**dev**->**status**,** **sizeof(**int**));**

**return** **sizeof(**int**);**

**}**

**}**

## 3.应用测试函数

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

int main**(**int argc**,** char **\*\***argv**)**

**{**

char ch**;**

int fd**,**fd0**,**fd1**,**fd2**;**

int tmp**;**

int tmpFd**;**

int val **=** 0**;**

char retChar**[**12**];**

fd **=** open**(**"/dev/leds"**,** O\_RDWR**);**

fd0 **=** open**(**"/dev/led0"**,** O\_RDWR**);**

fd1 **=** open**(**"/dev/led1"**,** O\_RDWR**);**

fd2 **=** open**(**"/dev/led2"**,** O\_RDWR**);**

printf**(**"JZ2440 leds control\n"**);**

printf**(**"1:all\n"**);**

printf**(**"2:led0\n"**);**

printf**(**"3:led1\n"**);**

printf**(**"4:led2\n"**);**

**while(** **(**ch**=**getchar**())** **!=** '\n'**)**

**{**

tmp **=** ch**-**48**;**

**}**

**if(**tmp **==** 1**)**

tmpFd **=** fd**;**

**else** **if(**tmp **==** 2**)**

tmpFd **=** fd0**;**

**else** **if(**tmp **==** 3**)**

tmpFd **=** fd1**;**

**else** **if(**tmp **==** 4**)**

tmpFd **=** fd2**;**

printf**(**"1.on\n"**);**

printf**(**"2.off\n"**);**

**while(** **(**ch**=**getchar**())** **!=** '\n'**)**

**{**

tmp **=** ch**-**48**;**

**}**

**if(**tmp **==** 1**)**

val **=** 1**;**

**else** **if(**tmp **==** 2**)**

val **=** 0**;**

//write(tmpFd, &val, 4);

ioctl**(**tmpFd**,** val **,**0**);**

int ret **=** read**(**tmpFd **,** retChar **,**12**);**

retChar**[**ret**]** **=** '\0'**;**

printf**(**"ret=%d\n"**,**ret**);**

**if(**ret **==** 12**)**

**{**

printf**(**"%d %d %d\n"**,\*(**int **\*)**retChar**,\*(**int **\*)(**retChar**+**4**),\*(**int **\*)(**retChar**+**8**));**

**}**

**else**

**{**

printf**(**"%d\n"**,\*(**int **\*)**retChar**);**

**}**

**return** 0**;**

**}**

## 4.总结

通过led驱动，了解了字符设备的基本写法，其中包含了寄存器动态方法和静态方法 获取，字符设备私有数据，一个主设备号多个次设备等概念。