# Linux驱动\_sys目录解析

Sysfs文件系统是一个类似于proc文件系统的特殊文件系统，用于将系统中的设备组织成层次结构，并向用户模式程序提供详细的内核[数据结构](http://lib.csdn.net/base/datastructure)信息。其实，就是在用户态可以通过对sys文件系统的访问，来看内核态的一些驱动或者设备等。

## 1.简单的led驱动sys目录解析

platform\_device和sys目录相关的代码

static struct platform\_device led\_platform\_device **=**

**{**

**.**name **=** "jz2440 led"**,**

**.**id **=** **-**1**,**

**.**num\_resources **=** ARRAY\_SIZE**(**led\_resource**),**

**.**resource **=** led\_resource**,**

**.**dev **=** **{**

**.**release **=** led\_platform\_device\_release**,**

**},**

**};**

static int \_\_init led\_platform\_device\_init**()**

**{**

**return** platform\_device\_register**(&**led\_platform\_device**);**

**}**

platform\_driver和sys目录相关的代码

static struct platform\_driver led\_platform\_driver **=**

**{**

**.**probe **=** led\_probe**,**

**.**remove **=** \_\_devexit\_p**(**led\_remove**),**

**.**driver **=** **{**

**.**name **=** "jz2440 led"**,**

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

**}**

**};**

static int \_\_init led\_platform\_driver\_init**()**

**{**

**return** platform\_driver\_register**(&**led\_platform\_driver**);**

**}**

static int \_\_devinit led\_probe**(**struct platform\_device **\***pdev**)**

**{**

led\_drv\_class **=** class\_create**(**THIS\_MODULE**,**”led”**);**

device\_create**(**led\_drv\_class**,** **NULL,** MKDEV**(**led\_major**,** 0**),** **NULL,**”led”**);**

**}**

platform device注册后生成的/sys目录如下：

**/**sys**/**devices**/**platform # ls **-**l

jz2440 led

**/**sys**/**devices**/**platform**/**jz2440 led # ls **-**l

driver **->** **../../../**bus**/**platform**/**drivers**/**jz2440 led

modalias

power

subsystem **->** **../../../**bus**/**platform

uevent

**/**sys**/**bus**/**platform**/**devices # ls **-**l

jz2440 led **->** **../../../**devices**/**platform**/**jz2440 led

platform driver注册后生成的/sys目录如下：

**/**sys**/**bus**/**platform**/**drivers**/**jz2440 led # ls **-**l

bind

jz2440 led **->** **../../../../**devices**/**platform**/**jz2440 led

module **->** **../../../../**module**/**led\_platform\_driver

uevent

unbind

device\_create后生成的/sys目录如下：

**/**sys**/**devices**/**virtual**/**led**/**led # ls **-**l

dev

power

subsystem **->** **../../../../**class**/**led

uevent

**/**sys**/**class**/**led # ls **-**l

led **->** **../../**devices**/**virtual**/**led**/**led

我们发现device\_create的时候parent为null，其实parent可以指定为platform\_device对应的dev，代码修改为：

device\_create**(**led\_drv\_class**,** **&**pdev**->**dev**,** MKDEV**(**led\_major**,** 0**),** **NULL,** DEVICE\_NAME**);**

**那么这样修改sys目录有什么变化呢？**

**/**sys**/**devices**/**platform**/**jz2440 led # ls **-**l

driver **->** **../../../**bus**/**platform**/**drivers**/**jz2440 led

**led**

modalias

power

subsystem **->** **../../../**bus**/**platform

uevent

红色部分为新增的目录，device\_create指定了parent，则可以认为led\_drv\_class的parent device是jz2440 led对应的platform\_device。所以led的目录出现在了jz2440 led platform\_device对应的sys目录中。**那么/sys/devices/virtual下就没有led目录了，/sys/class/led目录下的链接也会发生改变。**

**/**sys**/**class**/**led # ls **-**l

total 0

led **->** **../../**devices**/**platform**/**jz2440 led**/**led**/**led

**总结下：**可以看到platform\_device和device\_create均会创建device。platform\_device和platform\_driver的注册是挂在platform bus上。

## 2. device\_create\_file

device\_create\_file可以在/sys目录下创建调试文件，生成的目录在对应的device目录下。你可以在platform\_device对应的device下创建，也可以在device\_create对应的device下创建。

## 3. driver\_data

每个device都一个对应的driver\_data，也就是当出现了device结构体时，便可以获取到对应的私有数据driver\_data。

int dev\_set\_drvdata**(**struct device **\***dev**,** void **\***data**)**

void **\***dev\_get\_drvdata**(**const struct device **\***dev**)**

static inline void platform\_set\_drvdata**(**struct platform\_device **\***pdev**,** void **\***data**)**

static inline void **\***platform\_get\_drvdata**(**const struct platform\_device **\***pdev**)**

static inline void i2c\_set\_clientdata**(**struct i2c\_client **\***dev**,** void **\***data**)**

static inline void **\***i2c\_get\_clientdata**(**const struct i2c\_client **\***dev**)**

driver\_data怎么用呢？例如可以在device\_create\_file中的属性函数中时候使用。

static ssize\_t hello\_val\_store**(**struct device**\*** dev**,** struct device\_attribute**\*** attr**,** const char**\*** buf**,** size\_t count**)** **{**

struct xxx\_dev**\*** hdev **=** **(**struct xxx\_dev**\*)**dev\_get\_drvdata**(**dev**);**

**return** \_\_hello\_set\_val**(**hdev**,** buf**,** count**);**

**}**

可以在remove的时候释放掉对应的driver\_data。

static int \_\_devexit dw8250\_remove**(**struct platform\_device **\***pdev**)**

**{**

struct dw8250\_data **\***data **=** platform\_get\_drvdata**(**pdev**);**

serial8250\_unregister\_port**(**data**->**line**);**

**return** 0**;**

**}**

还有其他用途，后面会详细分析。

## 4. subsys\_system\_register

struct bus\_type s3c2440\_subsys **=** **{**

**.**name **=** "s3c2440-core"**,**

**.**dev\_name **=** "s3c2440-core"**,**

**};**

static int \_\_init s3c2440\_core\_init**(**void**)**

**{**

**return** subsys\_system\_register**(&**s3c2440\_subsys**,** **NULL);**

**}**

生成的sys目录内容为：

**/**sys**/**devices**/**system**/**s3c2440**-**core # ls **-**l

power

s3c2440**-**core0

uevent

**/**sys**/**bus**/**s3c2440**-**core # ls **-**l

total 0

devices

drivers

drivers\_autoprobe

drivers\_probe

uevent

## 5. leds子系统sys目录分析

leds子系统platform\_device内容：

static struct platform\_device smdk\_led4 **=** **{**

**.**name **=** "s3c24xx\_led"**,**

**.**id **=** 0**,**

**.**dev **=** **{**

**.**platform\_data **=** **&**smdk\_pdata\_led4**,**

**},**

**};**

static struct platform\_device smdk\_led5 **=** **{**

**.**name **=** "s3c24xx\_led"**,**

**.**id **=** 1**,**

**.**dev **=** **{**

**.**platform\_data **=** **&**smdk\_pdata\_led5**,**

**},**

**};**

static struct platform\_device smdk\_led6 **=** **{**

**.**name **=** "s3c24xx\_led"**,**

**.**id **=** 2**,**

**.**dev **=** **{**

**.**platform\_data **=** **&**smdk\_pdata\_led6**,**

**},**

**};**

static struct platform\_device smdk\_led7 **=** **{**

**.**name **=** "s3c24xx\_led"**,**

**.**id **=** 3**,**

**.**dev **=** **{**

**.**platform\_data **=** **&**smdk\_pdata\_led7**,**

**},**

**};**

platfform\_device注册后sys目录会生成如下内容：

**/**sys**/**devices**/**platform # ls **-**l

s3c24xx\_led.0

s3c24xx\_led.1

s3c24xx\_led.2

s3c24xx\_led.3

**/**sys**/**bus**/**platform**/**devices # ls **-**l

s3c24xx\_led.0 **->** **../../../**devices**/**platform**/**s3c24xx\_led.0

s3c24xx\_led.1 **->** **../../../**devices**/**platform**/**s3c24xx\_led.1

s3c24xx\_led.2 **->** **../../../**devices**/**platform**/**s3c24xx\_led.2

s3c24xx\_led.3 **->** **../../../**devices**/**platform**/**s3c24xx\_led.3

注意：s3c24xx\_led后面的数字为对应的id号。

paltform\_driver内容：

static struct platform\_driver s3c24xx\_led\_driver **=** **{**

**.**probe **=** s3c24xx\_led\_probe**,**

**.**remove **=** s3c24xx\_led\_remove**,**

**.**driver **=** **{**

**.**name **=** "s3c24xx\_led"**,**

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

**},**

**};**

platform\_driver注册后sys目录会出现如下内容：

**/**sys**/**bus**/**platform**/**drivers**/**s3c24xx\_led # ls **-**l

bind

module **->** **../../../../**module**/**leds\_s3c24xx

s3c24xx\_led.0 **->** **../../../../**devices**/**platform**/**s3c24xx\_led.0

s3c24xx\_led.1 **->** **../../../../**devices**/**platform**/**s3c24xx\_led.1

s3c24xx\_led.2 **->** **../../../../**devices**/**platform**/**s3c24xx\_led.2

s3c24xx\_led.3 **->** **../../../../**devices**/**platform**/**s3c24xx\_led.3

uevent

unbind

probe函数中调用了如下函数：

led\_classdev\_register**(&**dev**->**dev**,** **&**led**->**cdev**);**

led\_cdev**->**dev **=** device\_create**(**leds\_class**,** parent**,** 0**,** led\_cdev**,** "%s"**,** led\_cdev**->**name**);**

leds子系统初始化创建了leds\_class和属性文件：

static struct device\_attribute led\_class\_attrs**[]** **=** **{**

\_\_ATTR**(**brightness**,** 0644**,** led\_brightness\_show**,** led\_brightness\_store**),**

\_\_ATTR**(**max\_brightness**,** 0444**,** led\_max\_brightness\_show**,** **NULL),**

#ifdef CONFIG\_LEDS\_TRIGGERS

\_\_ATTR**(**trigger**,** 0644**,** led\_trigger\_show**,** led\_trigger\_store**),**

#endif

\_\_ATTR\_NULL**,**

**};**

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

**{**

leds\_class **=** class\_create**(**THIS\_MODULE**,** "leds"**);**

leds\_class**->**dev\_attrs **=** led\_class\_attrs**;**

**return** 0**;**

**}**

那么这些动作后sys目录会出现哪些文件呢？

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

led4 **->** **../../**devices**/**platform**/**s3c24xx\_led.0**/**leds**/**led4

led5 **->** **../../**devices**/**platform**/**s3c24xx\_led.1**/**leds**/**led5

led6 **->** **../../**devices**/**platform**/**s3c24xx\_led.2**/**leds**/**led6

led7 **->** **../../**devices**/**platform**/**s3c24xx\_led.3**/**leds**/**led7

**/**sys**/**class**/**leds # ls **-**l led4**/**

brightness

device **->** **../../../**s3c24xx\_led.0

max\_brightness

power

subsystem **->** **../../../../../**class**/**leds

trigger

uevent

因为parent指定为platform\_device对应的device，所以出现在了s3c24xx\_led.x目录中，另外每个设备文件下还出现了brightness等属性文件。

**leds子系统下platform\_data,drvdata等数据使用分析：**

**platform\_device对应的platform\_data：**

static struct s3c24xx\_led\_platdata smdk\_pdata\_led4 **=** **{**

**.**gpio **=** S3C2410\_GPF**(**4**),**

**.**flags **=** S3C24XX\_LEDF\_ACTLOW **|** S3C24XX\_LEDF\_TRISTATE**,**

**.**name **=** "led4"**,**

**.**def\_trigger **=** "timer"**,**

**};**

platform\_data用于描述设备和硬件相关的一些私有属性，方便实际操作的时候访问。

**platform\_device对应的driver\_data：**

一般在probe的过程中，会创建一个大的结构体：

struct s3c24xx\_gpio\_led **{**

struct led\_classdev cdev**;**

struct s3c24xx\_led\_platdata **\***pdata**;**

**};**

同时该结构体会作为platform\_deivce对应的driver\_data。

struct s3c24xx\_gpio\_led **\***led **=**

kzalloc**(sizeof(**struct s3c24xx\_gpio\_led**),** GFP\_KERNEL**);**

platform\_set\_drvdata**(**dev**,** led**);**

那么platform\_device对应的driver\_data在哪儿使用呢？

remove的时候做一些释放动作。

static int s3c24xx\_led\_remove**(**struct platform\_device **\***dev**)**

**{**

struct s3c24xx\_gpio\_led **\***led **=** pdev\_to\_gpio**(**dev**);**

led\_classdev\_unregister**(&**led**->**cdev**);**

kfree**(**led**);**

**return** 0**;**

**}**

**字符设备device对应的driver\_data：**

在创建led字符设备时，也为device指定了driver\_data。

led\_cdev**->**dev **=** device\_create**(**leds\_class**,** parent**,** 0**,** led\_cdev**,** "%s"**,** led\_cdev**->**name**);**

这里的driver\_data为led\_cdev，也就是上面s3c24xx\_gpio\_led中对应的led\_classdev。led\_classdev为leds子系统的通用结构体，定义了leds标准操作。

那么字符设备device对应的driver\_data在哪儿使用呢？

static ssize\_t led\_brightness\_show**(**struct device **\***dev**,** struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

struct led\_classdev **\***led\_cdev **=** dev\_get\_drvdata**(**dev**);**

led\_update\_brightness**(**led\_cdev**);**

**return** sprintf**(**buf**,** "%u\n"**,** led\_cdev**->**brightness**);**

**}**

## 6. probe时候创建的大结构体

struct large\_device

**{**

struct device dev**;** //device\_create对应的dev

struct cdev char\_dev**;** //cdev\_init对应的cdev

void **\*** private\_data**;** //私有数据

**}**

一般会将这个结构体作为platform\_deivce的driver\_data。

这样的好处是当platform\_device卸载的时候进行device\_destroy cdev\_del kfree等操作，当然你也可以使用全局变量。

同样的这样也方便open/device\_create\_file等操作，利用container\_of便可以得到larger\_device。

在open操作中，你可以借助cdev反向得到larger\_device并得到private\_data。

在device\_create\_file创建的属性文件中，可以借助dev反向得到large\_device。

## 7.2440 sys目录解析

# ls /sys/

block     class     firmware  kernel    power

bus       devices   fs        module

### 1.1 /sys/devices

# ls /sys/devices/

platform  system

# ls /sys/devices/platform/

dm9000.0        s3c2410**-**rtc     s3c2440**-**uart.0  s3c24xx\_led.2

power           s3c2410**-**wdt     s3c2440**-**uart.1  s3c24xx\_led.3

s3c2410**-**iis     s3c2440**-**i2c     s3c2440**-**uart.2  serial8250

s3c2410**-**lcd     s3c2440**-**nand    s3c24xx\_led.0   serial8250.0

s3c2410**-**ohci    s3c2440**-**sdi     s3c24xx\_led.1   uevent

# ls /sys/devices/system/

clocksource   s3c2410**-**core  s3c2440**-**core  s3c2443**-**core  timekeeping

cpu           s3c2412**-**core  s3c2442**-**core  s3c24xx**-**dma   timer

该目录下是全局设备结构体系，包含所有被发现的注册在各种总线上的各种物理设备。一般来说，所有的物理设备都按其在总线上的拓扑结构来显示，但有两个例外，即platform devices和system devices。platform devices一般是挂在芯片内部的高速或者低速总线上的各种控制器和外设，它们能被CPU直接寻址；system devices不是外设，而是芯片内部的核心结构，比如CPU，timer等，它们一般没有相关的驱动，但是会有一些体系结构相关的代码来配置它们。

### 1.2 /sys/dev

#ls /sys/dev

block char

# ls /sys/dev/char

10**:**1 10**:**59 1**:**3 13**:**75 29**:**0 4**:**23 4**:**39 4**:**54 4**:**7 4**:**85 7**:**1

10**:**183 10**:**60 13**:**0 13**:**76 4**:**0 4**:**24 4**:**4 4**:**55 4**:**70 4**:**86 7**:**128

10**:**184 10**:**61 13**:**1 1**:**4 4**:**1 4**:**25 4**:**40 4**:**56 4**:**71 4**:**87 7**:**129

10**:**200 10**:**62 13**:**32 1**:**5 4**:**10 4**:**26 4**:**41 4**:**57 4**:**72 4**:**88 7**:**130

10**:**223 10**:**63 13**:**33 1**:**7 4**:**11 4**:**27 4**:**42 4**:**58 4**:**73 4**:**89 7**:**131

10**:**227 108**:**0 13**:**63 1**:**8 4**:**12 4**:**28 4**:**43 4**:**59 4**:**74 4**:**9 7**:**132

10**:**228 1**:**1 13**:**64 189**:**0 4**:**13 4**:**29 4**:**44 4**:**6 4**:**75 4**:**90 7**:**133

10**:**229 1**:**11 13**:**65 189**:**1 4**:**14 4**:**3 4**:**45 4**:**60 4**:**76 4**:**91 7**:**134

ls **/**sys**/**dev**/**block**/**

1**:**0 1**:**10 1**:**11 1**:**13 1**:**15 1**:**3 1**:**5 1**:**7 1**:**9 7**:**1 7**:**3 7**:**5 7**:**7 8**:**1 8**:**5

1**:**1 11**:**0 1**:**12 1**:**14 1**:**2 1**:**4 1**:**6 1**:**8 7**:**0 7**:**2 7**:**4 7**:**6 8**:**0 8**:**2

lrwxrwxrwx 1 root root 0 Mar 10 09**:**42 4**:**81 **->** **../../**devices**/**platform**/**serial8250**/**tty**/**ttyS17

目录下维护一个按照字符设备和块设备的主次号码(major:minor)链接到真是设备(/sys/devices)的符号链接文件。

### 1.3. /sys/class

# ls /sys/class/

graphics      misc          printer       sound         vc

hwmon         mmc\_host      rtc           spi\_master    vtconsole

i2c**-**adapter   mtd           scsi\_device   tty

input         net           scsi\_disk     usb\_endpoint

mem           ppdev         scsi\_host     usb\_host

该目录下包含所有注册在kernel里面的设备类型，这是按照设备功能分类的设备模型，每个设备类型表达具有一种功能的设备。每个设备类型子目录下都是这种设备类型的各种具体设备的符号链接，这些链接指向/sys/devices/name下的具体设备。设备类型和设备并没有一一对应的关系，一个物理设备可能具备多种设备类型；一个设备类型只表达具有一种功能的设备，比如：系统所有输入设备都会出现在/sys/class/input之下，而不论它们是以何种总线连接到系统的。(/sys/class也是构成linux统一设备模型的一部分)。

4./sys/block

# ls /sys/block/

loop0      loop4      mtdblock0  ram0       ram12      ram2       ram6

loop1      loop5      mtdblock1  ram1       ram13      ram3       ram7

loop2      loop6      mtdblock2  ram10      ram14      ram4       ram8

loop3      loop7      mtdblock3  ram11      ram15      ram5       ram9

该目录下的所有子目录代表着系统中当前被发现的所有块设备。按照功能来说放置在/sys/class下会更合适，但由于历史遗留因素而一直存在于/sys/block，但从linux2.6.22内核开始这部分就已经标记为过去时，只有打开了CONFIG\_SYSFS\_DEPRECATED配置编译才会有这个目录存在，并且其中的内容在从linux2.6.26版本开始已经正式移到了/sys/class/block，旧的接口/sys/block为了向后兼容而保留存在，但其中的内容已经变为了指向它们在/sys/devices/中真实设备的符号链接文件。

5./sys/bus

# ls /sys/bus/

i2c       mmc       platform  scsi      serio     spi       usb

该目录下的每个子目录都是kernel支持并且已经注册了的总线类型。这是内核设备按照总线类型分层放置的目录结构，/sys/devices中的所有设备都是连接于某种总线之下的，bus子目录下的每种具体总线之下可以找到每个具体设备的符号链接，一般来说每个子目录(总线类型)下包含两个子目录，一个是devices，另一个是drivers；其中devices下是这个总线类型下的所有设备，这些设备都是符号链接，它们分别指向真正的设备(/sys/devices/name/下)；而drivers下是所有注册在这个总线上的驱动，每个driver子目录下 是一些可以观察和修改的driver参数。

6./sys/fs

【空】

按照设计，该目录使用来描述系统中所有的文件系统，包括文件系统本身和按照文件系统分类存放的已挂载点。

7./sys/kernel

# ls /sys/kernel/

uevent\_helper  uevent\_seqnum

这个目录下存放的是内核中所有可调整的参数

8./sys/firmware

这里是系统加载固件机制的对用户空间的接口，关于固件有专用于固件加载的一套API，在附录 LDD3 一书中有关于内核支持固件加载机制的更详细的介绍；

9./sys/module

# ls /sys/module/

8250          loop          rcupdate      snd\_pcm       usb\_storage

atkbd         lp            rd            snd\_soc\_core  usbcore

dm9dev9000c   mousedev      s3c2410\_wdt   snd\_timer     usbhid

hid           nfs           s3c\_ts        sunrpc        vt

keyboard      ohci\_hcd      scsi\_mod      tcp\_cubic     yaffs

lockd         printk        snd           tsdev

该目录下有系统中所有的模块信息，不论这些模块是以内联(inlined)方式编译到内核映像文件中还是编译为外模块(.ko文件)，都可能出现在/sys/module中。即module目录下包含了所有的被载入kernel的模块。

10./sys/power

\# ls /sys/power/

state

该目录是系统中的电源选项，对正在使用的power子系统的描述。这个目录下有几个属性文件可以用于控制整个机器的电源状态，如可以向其中写入控制命令让机器关机/重启等等。

多个device 一个driver

device\_register函数

device\_create\_file在probe的时候创建，以pdev-dev为节点和device\_create后的节点有什么不一样

device\_create parent指针

分析每个目录的内容和链接，每个目录和文件是如何创建的，

写一个最简单的led程序也要分析下char设备的sys目录

sys\_create\_file生成的文件在哪儿 /sys/class /sys/devices

分析每个子系统的sys下文件的构成，将以前的驱动代码挂上去再分析下。

device\_create**(**class**,** **NULL,** MKDEV**(**major**,** 0**),** **NULL,** "at24cxx"**);**

/sys/class/at24cxx指向哪儿？/sys/devices/?

写一个led驱动，platform方式，probe中device\_create 指定parent为null或pdev-dev，然后生成的sys有什么不一样

platform\_data driver\_data

platform\_device platform\_driver

--->probe

--->device\_create sys/class/xxx 这个指向platform的哪儿呢？ 如果有parent呢？

platform\_device和device\_create中对应的device有什么不同

static struct device\_attribute rtc\_attrs**[]** **=** **{**

\_\_ATTR**(**name**,** S\_IRUGO**,** rtc\_sysfs\_show\_name**,** **NULL),**

\_\_ATTR**(**date**,** S\_IRUGO**,** rtc\_sysfs\_show\_date**,** **NULL),**

\_\_ATTR**(**time**,** S\_IRUGO**,** rtc\_sysfs\_show\_time**,** **NULL),**

\_\_ATTR**(**since\_epoch**,** S\_IRUGO**,** rtc\_sysfs\_show\_since\_epoch**,** **NULL),**

**{** **},**

**};**

void \_\_init rtc\_sysfs\_init**(**struct class **\***rtc\_class**)**

**{**

rtc\_class**->**dev\_attrs **=** rtc\_attrs**;**

**}**