Linux Platform Device and Driver

从Linux 2.6起引入了一套新的驱动管理和注册机制:Platform_device和Platform_driver。Linux中大部分的设备驱动，都可以使用这套机制, 设备用Platform_device表示，驱动用Platform_driver进行注册。

Linux platform driver机制和传统的device driver 机制(通过driver_register函数进行注册)相比，一个十分明显的优势在于platform机制将设备本身的资源注册进内核，由内核统一管理，在驱动程序中使用这些资源时通过platform device提供的标准接口进行申请并使用。这样提高了驱动和资源管理的独立性，并且拥有较好的可移植性和安全性(这些标准接口是安全的)。

Platform机制的本身使用并不复杂，由两部分组成：platform_device和platfrom_driver。通过Platform机制开发发底层驱动的大致流程为: 定义 platform_device à 注册 platform_device à 定义 platform_driver à注册 platform_driver。

首先要确认的就是设备的资源信息，例如设备的地址，中断号等。
在2.6内核中platform设备用结构体platform_device来描述，该结构体定义在kernel\include\linux\platform_device.h中，

```
struct platform_device {
const char * name;
u32   id;
struct device dev;
u32   num_resources;
struct resource * resource;
};
```

该结构一个重要的元素是resource，该元素存入了最为重要的设备资源信息，定义在kernel\include\linux\ioport.h中，

```
struct resource {
const char *name;
unsigned long start, end;
unsigned long flags;
struct resource *parent, *sibling, *child;
};
```

下面举s3c2410平台的i2c驱动作为例子来说明：

```
/* arch/arm/mach-s3c2410/devs.c */
/* I2C */
static struct resource s3c_i2c_resource[] = {
     [0] = {
          .start = S3C24XX_PA_IIC,
          .end = S3C24XX_PA_IIC + S3C24XX_SZ_IIC - 1,
          .flags = IORESOURCE_MEM,
     },
```

```
    [1] = {
        .start = IRQ_IIC, //S3C2410_IRQ(27)
        .end = IRQ_IIC,
        .flags = IORESOURCE_IRQ,
    }
};
```

这里定义了两组resource，它描述了一个I2C设备的资源，第1组描述了这个I2C设备所占用的总线地址范围，IORESOURCE_MEM表示第1组描述的是内存类型的资源信息，第2组描述了这个I2C设备的中断号，IORESOURCE_IRQ表示第2组描述的是中断资源信息。设备驱动会根据flags来获取相应的资源信息。

有了resource信息，就可以定义platform_device了：

```
struct platform_device s3c_device_i2c = {
    .name = "s3c2410-i2c",
    .id = -1,
    .num_resources = ARRAY_SIZE(s3c_i2c_resource),
    .resource = s3c_i2c_resource,
};
```

定义好了platform_device结构体后就可以调用函数platform_add_devices向系统中添加该设备了，之后可以调用platform_driver_register()进行设备注册。要注意的是，这里的platform_device设备的注册过程必须在相应设备驱动加载之前被调用，即执行platform_driver_register之前,原因是因为驱动注册时需要匹配内核中所以已注册的设备名。

s3c2410-i2c的platform_device是在系统启动时，在cpu.c里的s3c_arch_init()函数里进行注册的，这个函数申明为arch_initcall(s3c_arch_init);会在系统初始化阶段被调用。arch_initcall的优先级高于module_init。所以会在Platform驱动注册之前调用。(详细参考include/linux/init.h)

s3c_arch_init函数如下：

```
/* arch/arm/mach-3sc2410/cpu.c */
static int __init s3c_arch_init(void)
{
int ret;
……
/* 这里board指针指向在mach-smdk2410.c里的定义的smdk2410_board，里面包含了预先定义的I2C Platform_device等. */
if (board != NULL) {
      struct platform_device **ptr = board->devices;
      int i;
      for (i = 0; i   board->devices_count; i++, ptr++) {
        ret = platform_device_register(*ptr); //在这里进行注册
        if (ret) {
          printk(KERN_ERR "s3c24xx: failed to add board device %s (%d) @%p\n", (*ptr)->name,
```

```
ret, *ptr);
          }
        }
        /* mask any error, we may not need all these board
       * devices */
        ret = 0;
}
return ret;
}
```

同时被注册还有很多其他平台的platform_device，详细查看arch/arm/mach-s3c2410/mach-smdk2410.c里的smdk2410_devices结构体。

驱动程序需要实现结构体struct platform_driver，参考drivers/i2c/busses

```
/* device driver for platform bus bits */
static struct platform_driver s3c2410_i2c_driver = {
      .probe = s3c24xx_i2c_probe,
      .remove = s3c24xx_i2c_remove,
      .resume = s3c24xx_i2c_resume,
      .driver = {
              .owner = THIS_MODULE,
              .name = "s3c2410-i2c",
      },
};
```

在驱动初始化函数中调用函数platform_driver_register()注册platform_driver，需要注意的是s3c_device_i2c结构中name元素和s3c2410_i2c_driver结构中driver.name必须是相同的，这样在platform_driver_register()注册时会对所有已注册的所有platform_device中的name和当前注册的platform_driver的driver.name进行比较，只有找到相同的名称的platfomr_device才能注册成功，当注册成功时会调用platform_driver结构元素probe函数指针，这里就是s3c24xx_i2c_probe,当进入probe函数后，需要获取设备的资源信息，常用获取资源的函数主要是：

```
struct resource * platform_get_resource(struct platform_device *dev, unsigned int type, unsigned int num);
```

根据参数type所指定类型，例如IORESOURCE_MEM，来获取指定的资源。

```
struct int platform_get_irq(struct platform_device *dev, unsigned int num);
```

获取资源中的中断号。

下面举s3c24xx_i2c_probe函数分析,看看这些接口是怎么用的。

前面已经讲了，s3c2410_i2c_driver注册成功后会调用s3c24xx_i2c_probe执行，下面看代码：

```
/* drivers/i2c/busses/i2c-s3c2410.c */
```

```
static int s3c24xx_i2c_probe(struct platform_device *pdev)
{
struct s3c24xx_i2c *i2c = &s3c24xx_i2c;
struct resource *res;
int ret;

/* find the clock and enable it */

i2c->dev = &pdev->dev;
i2c->clk = clk_get(&pdev->dev, "i2c");
if (IS_ERR(i2c->clk)) {
    dev_err(&pdev->dev, "cannot get clock\n");
    ret = -ENOENT;
    goto out;
}
dev_dbg(&pdev->dev, "clock source %p\n", i2c->clk);
clk_enable(i2c->clk);
/* map the registers */
res = platform_get_resource(pdev, IORESOURCE_MEM, 0); /* 获取设备的IO资源地址 */
if (res == NULL) {
    dev_err(&pdev->dev, "cannot find IO resource\n");
    ret = -ENOENT;
    goto out;
}

i2c->ioarea = request_mem_region(res->start, (res->end-res->start)+1, pdev->name);
/* 申请这块IO Region */

if (i2c->ioarea == NULL) {
    dev_err(&pdev->dev, "cannot request IO\n");
    ret = -ENXIO;
    goto out;
}

i2c->regs = ioremap(res->start, (res->end-res->start)+1); /* 映射至内核虚拟空间 */

if (i2c->regs == NULL) {
    dev_err(&pdev->dev, "cannot map IO\n");
    ret = -ENXIO;
    goto out;
}

dev_dbg(&pdev->dev, "registers %p (%p, %p)\n", i2c->regs, i2c->ioarea, res);

/* setup info block for the i2c core */
i2c->adap.algo_data = i2c;
```

```
i2c->adap.dev.parent = &pdev->dev;

/* initialise the i2c controller */
ret = s3c24xx_i2c_init(i2c);
if (ret != 0)
    goto out;
/* find the IRQ for this unit (note, this relies on the init call to ensure no current IRQs
pending */

res = platform_get_resource(pdev, IORESOURCE_IRQ, 0); /* 获取设备IRQ中断号 */
if (res == NULL) {
    dev_err(&pdev->dev, "cannot find IRQ\n");
    ret = -ENOENT;
    goto out;
}

ret = request_irq(res->start, s3c24xx_i2c_irq, IRQF_DISABLED, /* 申请IRQ */
    pdev->name, i2c);

......
return ret;

}
```

小思考：
那什么情况可以使用platform driver机制编写驱动呢？
我的理解是只要和内核本身运行依赖性不大的外围设备(换句话说只要不在内核运行所需的一个最小系统之内的设备),相对独立的,拥有各自独自的资源(addresses and IRQs)，都可以用platform_driver实现。如：lcd,usb,uart等，都可以用platfrom_driver写，而timer,irq等最小系统之内的设备则最好不用platfrom_driver机制，实际上内核实现也是这样的。

参考资料：
linux-2.6.24/Documentation/driver-model/platform.txt

《platform _device和platform_driver注册过程》
http://blog.chinaunix.net/u2/60011/showart.php?id=1018999

http://www.eetop.cn/blog/html/45/11145-676.html