

## 实验 07 驱动注册

### 7.1 本章导读

前面的实验介绍过注册驱动的流程，注册的时候需要和设备匹配，简单介绍了将驱动注册到平台设备的结构体“platform\_driver\_register”，并没有介绍怎么使用使用。本章就个大家详细介绍如何使用这个结构体来注册驱动。

#### 7.1.1 工具

##### 7.1.1.1 硬件工具

- 1 ) iTOP4412 开发板
- 2 ) U 盘或者 TF 卡
- 3 ) PC 机
- 4 ) 串口

##### 7.1.1.2 软件工具

- 1 ) 虚拟机 Vmware
- 2 ) Ubuntu12.04.2
- 3 ) 超级终端（串口助手）
- 4 ) 实验配套源码文件夹“probe\_linux\_module”

## 7.1.2 预备课程

## 7.1.3 视频和代码资源

本节配套视频为 “视频 07\_驱动注册”

源码为 “probe\_linux\_module”

## 7.2 学习目标

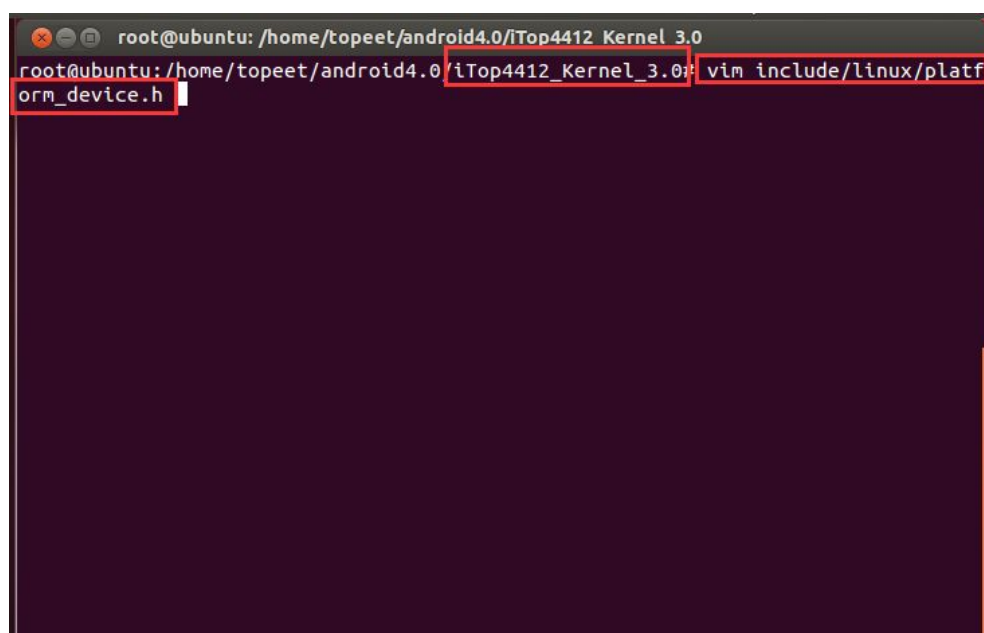
本章需要学习以下内容：

掌握将驱动注册到虚拟平台总线的方法

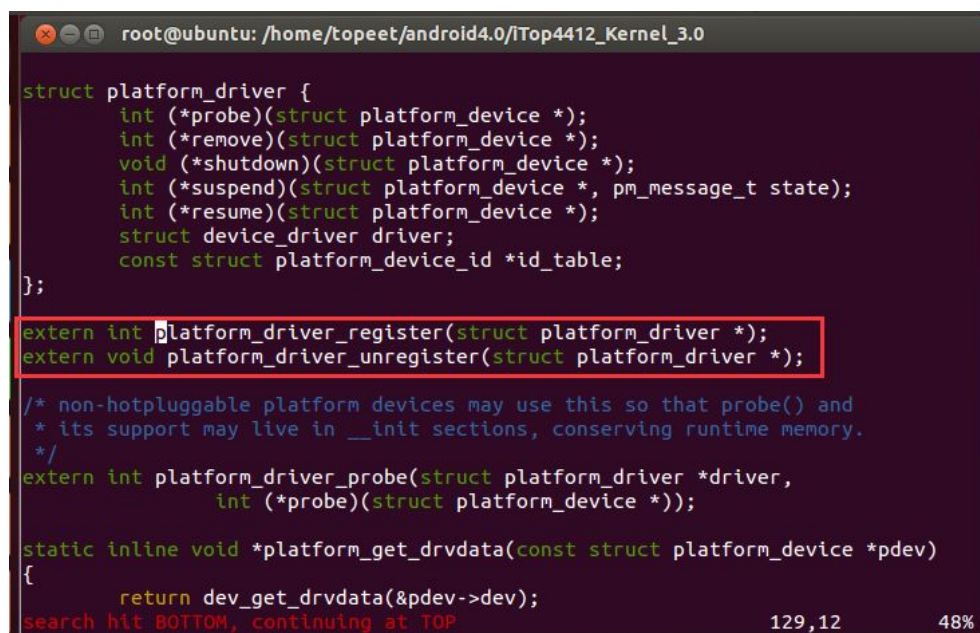
## 7.3 platform\_driver\_register 和 platform\_driver\_unregister 函数

platform\_driver\_register 函数和 platform\_driver\_unregister 函数用于注册和卸载驱动。

如下图所示，在 Linux 源码目录下，使用命令 “vim include/linux/platform\_device.h”。



查找一下 “platform\_driver\_register” ，如下图所示。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0

struct platform_driver {
    int (*probe)(struct platform_device *);
    int (*remove)(struct platform_device *);
    void (*shutdown)(struct platform_device *);
    int (*suspend)(struct platform_device *, pm_message_t state);
    int (*resume)(struct platform_device *);
    struct device_driver driver;
    const struct platform_device_id *id_table;
};

extern int platform_driver_register(struct platform_driver *);
extern void platform_driver_unregister(struct platform_driver *);

/* non-hotpluggable platform devices may use this so that probe() and
 * its support may live in __init sections, conserving runtime memory.
 */
extern int platform_driver_probe(struct platform_driver *driver,
    int (*probe)(struct platform_device *));

static inline void *platform_get_drvdata(const struct platform_device *pdev)
{
    return dev_get_drvdata(&pdev->dev);
}
search hit BOTTOM, continuing at TOP 129,12 48%
```

如上图所示。

注册驱动函数：

```
extern int platform_driver_register(struct platform_driver *)
```

卸载驱动函数：

```
extern void platform_driver_unregister(struct platform_driver *)
```

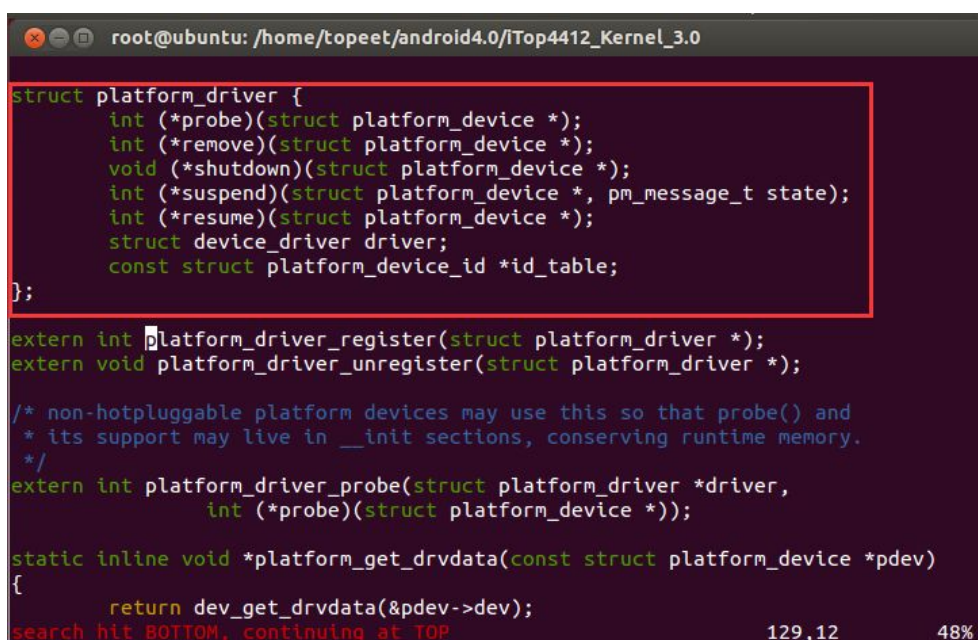
具体这两个函数是怎么实现的，大家其实不用去关心，都是前人做好的，只要掌握如何调用即可。

这两个函数都会调用一个 platform\_driver 类型的结构体。

## 7.4 platform\_driver 结构体

前一小节介绍到注册和卸载驱动的函数需要调用 platform\_driver 类型结构体。这个结构体非常重要，大家一定要掌握它的使用方法。

如下图所示，这个结构体也是定义在 “include/linux/platform\_device.h” 头文件中。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0

struct platform_driver {
    int (*probe)(struct platform_device *);
    int (*remove)(struct platform_device *);
    void (*shutdown)(struct platform_device *);
    int (*suspend)(struct platform_device *, pm_message_t state);
    int (*resume)(struct platform_device *);
    struct device_driver driver;
    const struct platform_device_id *id_table;
};

extern int platform_driver_register(struct platform_driver *);
extern void platform_driver_unregister(struct platform_driver *);

/* non-hotpluggable platform devices may use this so that probe() and
 * its support may live in __init sections, conserving runtime memory.
 */
extern int platform_driver_probe(struct platform_driver *driver,
    int (*probe)(struct platform_device *));

static inline void *platform_get_drvdata(const struct platform_device *pdev)
{
    return dev_get_drvdata(&pdev->dev);
}
search hit BOTTOM, continuing at TOP 129,12 48%
```

这个结构体需要好好分析，这个结构体是非常非常的重要，几乎所有的驱动都会用到。

该结构中包含了一组操作函数和一个 struct device\_driver 的对象。在驱动中首先要做的就是定义 platform\_driver 中的函数，并创建这个结构的一个对象实例，然后在 init() 函数中调用 platform\_driver\_register() 向系统注册驱动。

函数 int (\*probe)(struct platform\_device \*);

主要是进行设备的探测和初始化。例如想调用一个 GPIO，那么首先需要探测这个 GPIO 是否被占用了，如果被占用了那么初始化失败，驱动注册也就失败了；如果没有被占用，那么就申明要占用它。

该函数中一般还会添加生成设备节点的函数，如果初始化成功，那么就会需要添加设备节点。设备节点的知识在下一节介绍。

函数 `int (*remove)(struct platform_device *)`;

移除驱动，该函数中一般用于去掉设备节点或者释放软硬件资源。

接着的三个函数：

`void (*shutdown)(struct platform_device *)`;

`int (*suspend)(struct platform_device *, pm_message_t state)`;

`int (*resume)(struct platform_device *)`;

从字面上就很好理解了，关闭驱动，悬挂（休眠）驱动以及恢复的时候该驱动要做什么。

接着的结构体 `struct device_driver driver`;

主要包含两个参数，一个是 `name` 参数，驱动名称（需要和设备驱动结构体中的 `name` 参数一样）；一个是 `owner`，一般是 `THIS_MODULE`。

下面给接着给大家介绍一个小知识点，以 `platform_driver` 结构体中的参数 `probe` 为例，这个参数指向 `platform_driver_probe` 函数，如下图所示。

```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0

struct platform_driver {
    int (*probe)(struct platform_device *);
    int (*remove)(struct platform_device *);
    void (*shutdown)(struct platform_device *);
    int (*suspend)(struct platform_device *, pm_message_t state);
    int (*resume)(struct platform_device *);
    struct device_driver driver;
    const struct platform_device_id *id_table;
};

extern int platform_driver_register(struct platform_driver *);
extern void platform_driver_unregister(struct platform_driver *);

/* non-hotpluggable platform devices may use this so that probe() and
 * its support may live in __init sections, conserving runtime memory.
 */
extern int platform_driver_probe(struct platform_driver *driver,
                                int (*probe)(struct platform_device *));

static inline void *platform_get_drvdata(const struct platform_device *pdev)
{
    return dev_get_drvdata(&pdev->dev);
}
search hit BOTTOM, continuing at TOP                                129,12    48%
```

在视频中没有介绍这个参数 platform\_driver 是从哪里传来的。

大家打开平台文件，看一下注册设备的代码，如下图所示，是 HELLO\_CTL 的结构体。

```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0

    .id          = -1,
};
#endif

#ifdef CONFIG_HELLO_CTL
struct platform_device s3c_device_hello_ctl = {
    .name = "hello_ctl",
    .id   = -1,
};
#endif

#ifdef CONFIG_LEDS_CTL
struct platform_device s3c_device_leds_ctl = {
    .name = "leds",
    .id   = -1,
};
#endif

#ifdef CONFIG_BUZZER_CTL
struct platform_device s3c_device_buzzer_ctl = {
    .name = "buzzer_ctl",
    .id   = -1,
};
};
2527,0-1    53%
```

上图红色的结构体和 platform\_driver\_probe 函数的第一个参数类型一样，如果你注册一个 HELLO\_CTL 驱动，那么你的初始化函数 platform\_driver\_probe 就会调用这个结构体。



虽然这个结构体在这里很简单，但是在后面专门的移植课程中，你会发现移植中有部分工作是解决这个结构体中的问题，另一部分工作是调试 bug。在具体使用的时候再详细介绍相关知识，这里只要知道注册驱动的时候会调用平台文件中对应结构体即可。

## 7.5 实验操作

在前面“实验 2mini\_linux\_module”的基础上添加注册驱动的代码。

如下图所示，是“mini\_linux\_module.c”的代码。

```
1  #include <linux/init.h>
2  #include <linux/module.h>
3
4  MODULE_LICENSE("Dual BSD/GPL");
5  MODULE_AUTHOR("TOPEET");
6
7  static int hello_init(void)
8  {
9      printk(KERN_EMERG "HELLO WORLD enter!\n");
10     return 0;
11 }
12
13 static void hello_exit(void)
14 {
15     printk(KERN_EMERG "HELLO WORLD exit!\n");
16 }
17
18
19 module_init(hello_init);
20 module_exit(hello_exit);
21
```

先将这个文件名改为 probe\_linux\_module.c。

然后在 probe\_linux\_module.c 文件中开始修改和添加带代码。

如下图所示，首先需要添加头文件“#include <linux/platform\_device.h>”，然后定义一个宏变量 DRIVER\_NAME，定义为“hello\_ctl”，需要和前面注册的 hello 设备的名称相同。

```
#include <linux/init.h>
#include <linux/module.h>

/*驱动注册的头文件，包含驱动的结构体和注册和卸载的函数*/
#include <linux/platform_device.h>

#define DRIVER_NAME "hello_ctl"

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("TOPEET");
```

然后在模块入口和出口调用函数 platform\_driver\_register 和 platform\_driver\_unregister，如下图所示，先将参数名定义为“&hello\_driver”。另外注册驱动的时候，会返回数值，将其打印出来判断是否注册成功。

```
static int hello_init(void)
{
    int DriverState;

    printk(KERN_EMERG "HELLO WORLD enter!\n");
    DriverState = platform_driver_register(&hello_driver);

    printk(KERN_EMERG "\tDriverState is %d\n", DriverState);
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_EMERG "HELLO WORLD exit!\n");

    platform_driver_unregister(&hello_driver);
}

module_init(hello_init);
module_exit(hello_exit);
```

如下图所示，定义结构体“hello\_driver”。

driver 中的 name 参数就是驱动名称，这里将前面定义的宏变量 DRIVER\_NAME 赋给它；另外一个参数 owner 一般默认为 THIS\_MODULE。



```
struct platform_driver hello_driver = {  
    .probe = hello_probe,  
    .remove = hello_remove,  
    .shutdown = hello_shutdown,  
    .suspend = hello_suspend,  
    .resume = hello_resume,  
    .driver = {  
        .name = DRIVER_NAME,  
        .owner = THIS_MODULE,  
    }  
};
```

然后定义函数 hello\_probe、hello\_remove、hello\_shutdown、hello\_suspend、hello\_resume。

```
static int hello_probe(struct platform_device *pdv){  
    printk(KERN_EMERG "\tinitialized\n");  
    return 0;  
}  
  
static int hello_remove(struct platform_device *pdv){  
    return 0;  
}  
  
static void hello_shutdown(struct platform_device *pdv){  
    ;  
}  
  
static int hello_suspend(struct platform_device *pdv){  
    return 0;  
}  
  
static int hello_resume(struct platform_device *pdv){  
    return 0;  
}
```

如上图所示，这里在 hello\_probe 函数中添加打印信息 “printk(KERN\_EMERG “\tinitialized\n”);”

如果设备和驱动匹配成功就会进入函数 hello\_probe 打印 “initialized” 。

接着需要修改一下 Makefile 文件，将 “mini\_linux\_module.o” 改为

“probe\_linux\_module.o”，注释部分用户可以自己修改，如下图所示。

```
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#这里是编译itop4412_hello.c这个文件编译成中间文件itop4412_hello.o
obj-m += probe_linux_module.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -C就是指调用执行的路径
#$(KDIR) Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD) 当前目录变量
#modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.o
```

在 Ubuntu 中的目录 “/home/topeet” 中新建目录 “probe\_linux\_module”，拷贝驱动文件 “probe\_linux\_module.c” 和编译文件 “Makefile” 到新建中，如下图所示。

```
root@ubuntu: /home/topeet
root@ubuntu:~# cd /home/topeet/
root@ubuntu:/home/topeet# mkdir probe_linux_module/
root@ubuntu:/home/topeet# ls probe_linux_module/
Makefile probe_linux_module.c
root@ubuntu:/home/topeet#
```

进入 “probe\_linux\_module” 目录，使用命令 “make” 编译 “probe\_linux\_module.c” ，如下图所示，生成模块文件 “probe\_linux\_module.ko” 。

```
root@ubuntu: /home/topeet/probe_linux_module
root@ubuntu:~# cd /home/topeet/
root@ubuntu:/home/topeet# mkdir probe_linux_module/
root@ubuntu:/home/topeet# ls probe_linux_module/
Makefile probe_linux_module.c
root@ubuntu:/home/topeet# cd probe_linux_module/
root@ubuntu:/home/topeet/probe_linux_module# make
make -C /home/topeet/android4.0/iTop4412_Kernel_3.0 M=/home/topeet/probe_linux_module modules
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
CC [M] /home/topeet/probe_linux_module/probe_linux_module.o
/home/topeet/probe_linux_module/probe_linux_module.c:43: warning: initialization from incompatible pointer type
Building modules, stage 2.
MODPOST 1 modules
CC /home/topeet/probe_linux_module/probe_linux_module.mod.o
LD [M] /home/topeet/probe_linux_module/probe_linux_module.ko
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
root@ubuntu:/home/topeet/probe_linux_module# ls
Makefile probe_linux_module.c probe_linux_module.mod.o
modules.order probe_linux_module.ko probe_linux_module.o
Module.symvers probe_linux_module.mod.c
root@ubuntu:/home/topeet/probe_linux_module#
```

启动开发板，拷贝 “probe\_linux\_module.ko” 到 U 盘，将 U 盘插入开发板，加载驱动文件 “probe\_linux\_module.ko”，如下图所示，可以看到打印出了 “initialized”，表明进入了 probe 函数。

```
COM1

[root@iTOP-4412]#
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]# [ 44.691922] CPU3: shutdown

[root@iTOP-4412]# insmod /mnt/udisk/probe linux module.ko
[ 52.576354] HELLO WORLD enter!
[ 52.578004] initialized
[ 52.581974] DriverState is 0
[root@iTOP-4412]#
```

使用命令 “rmmod probe\_linux\_module” 卸载驱动，可以看到打印 “HELLO WORLD exit!”，表明卸载成功。

```
[root@iTOP-4412]#
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]# [ 44.691922] CPU3: shutdown

[root@iTOP-4412]# insmod /mnt/udisk/probe_linux_module.ko
[ 52.576354] HELLO WORLD enter!
[ 52.578004] initialized
[ 52.581974] DriverState is 0
[root@iTOP-4412]# lsmod
probe_linux_module 1244 0 - Live 0xbf000000
[root@iTOP-4412]# rmmod probe_linux_module
[ 116.633719] HELLO WORLD exit!
[root@iTOP-4412]#
```