linux设备驱动的总线驱动设备模型

一.主要原理

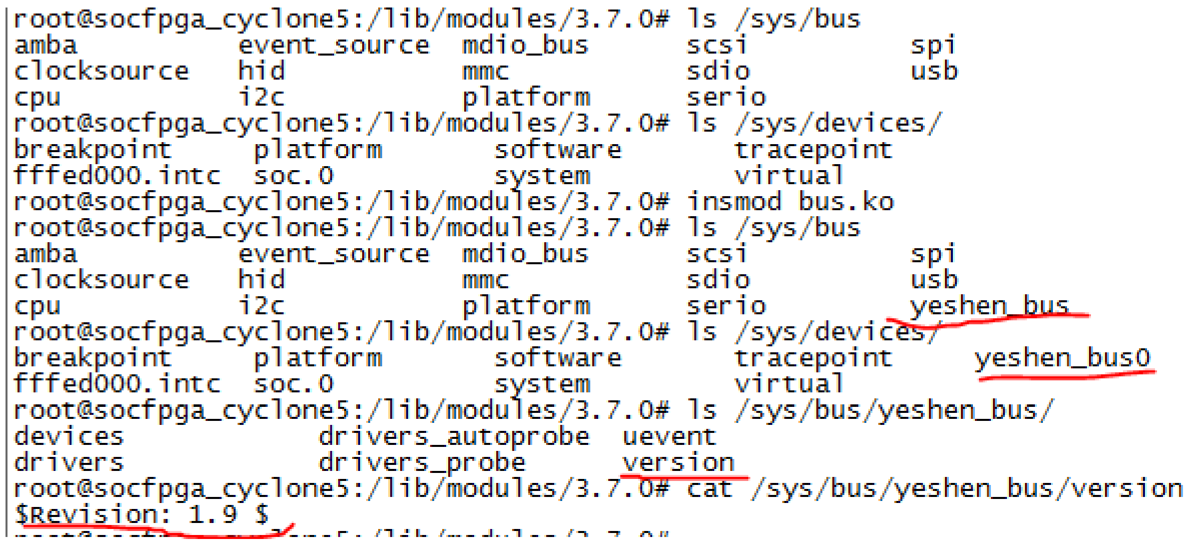
在linux内核中，分别使用bus_type、device_driver、device三个结构体表示总线、驱动和设备。其中对应的驱动和设备的device_driver和device都包含指向相同总线的bus_type指针。设备和驱动是分开注册的，每当设备或者驱动注册时都会调用bus_type中的match函数匹配，如果匹配成功则调用device_driver的probe初始化函数。总线、驱动和设备会落实为sysfs文件系统中的一个目录，而它们的attribute会落实为目录中的文件，attribute会伴随show()和store()两个函数，分别在读和写attribute对应的sysfs文件结点时候调用。

二.示例代码

1.实现一条总线

```
#include <linux/device.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/string.h>

MODULE_LICENSE("Dual BSD/GPL");

static char *Version = "$Revision: 1.9 $";

/* 驱动和设备匹配 */
static int my_match(struct device *dev, struct device_driver *driver)
{
    //return !strncmp(dev_name(dev), driver->name, strlen(driver->name));
    int ret;
    ret = !strncmp(dev_name(dev), driver->name, strlen(driver->name));
    if (ret)
        printk("driver and device match\n");

    return ret;
}



static void my_bus_release(struct device *dev)
{
    printk(KERN_DEBUG "my bus release\n");
}

/* 总线也是一个设备 */
struct device my_bus = {
    .init_name = "yeshen_bus0",
    .release   = my_bus_release
};

/* 定义一个总线 */
struct bus_type my_bus_type = {
    .name = "yeshen_bus",
    .match = my_match,
};

/* 导出总线和总线设备 */
EXPORT_SYMBOL(my_bus);
EXPORT_SYMBOL(my_bus_type);


/*
 * Export a simple attribute.
 */
static ssize_t show_bus_version(struct bus_type *bus, char *buf)
{
    return snprintf(buf, PAGE_SIZE, "%s\n", Version);
}

/* 定义了bus_attr_version属性 */
static BUS_ATTR(version, S_IRUGO, show_bus_version, NULL);


static int __init my_bus_init(void)
{
    int ret;

    /*注册总线*/
    ret = bus_register(&my_bus_type);
    if (ret)
        return ret;

    /*在yeshen_bus目录下创建属性文件*/
    if (bus_create_file(&my_bus_type, &bus_attr_version))
        printk(KERN_NOTICE "Fail to create version attribute!\n");

    /*注册总线设备,可以sys/devices/目录下面看到*/
    ret = device_register(&my_bus);
    if (ret)
        printk(KERN_NOTICE "Fail to register device:my_bus!\n");

    return ret;
}

static void my_bus_exit(void)
{
    device_unregister(&my_bus);
    bus_unregister(&my_bus_type);
}
```

```
module_init(my_bus_init);
module_exit(my_bus_exit);
```

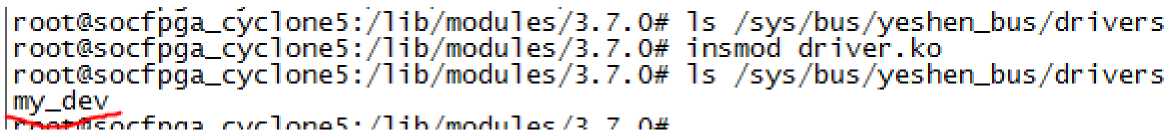


```
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus
amba           event_source   mdio_bus       scsi           spi
clocksource    hid            mmc            sdio           usb
cpu            i2c            platform       serio
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/devices/
breakpoint      platform       software       tracepoint
fffed000.intc   soc.0          system         virtual
root@socfpga_cyclone5:/lib/modules/3.7.0# insmod bus.ko
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus
amba           event_source   mdio_bus       scsi           spi
clocksource    hid            mmc            sdio           usb
cpu            i2c            platform       serio          yeshen_bus
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/devices/
breakpoint      platform       software       tracepoint     yeshen_bus0
fffed000.intc   soc.0          system         virtual
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus/yeshen_bus/
devices             drivers_autoprobe  uevent
drivers             drivers_probe      version
root@socfpga_cyclone5:/lib/modules/3.7.0# cat /sys/bus/yeshen_bus/version
$Revision: 1.9 $
```

程序中分别以总线和设备将这条总线注册，因此在/sys/bus和/sys/devices中都出现了对应的名字。并且在/sys/bus/yeshen_bus目录下创建属性文件version，使用cat读取该文件会调用绑定的show_bus_version函数打印出$Revision: 1.9 $。

2.实现一个挂接在上面总线的驱动

```
#include <linux/device.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/string.h>


MODULE_LICENSE("Dual BSD/GPL");


/* 使用导出的总线 */
extern struct bus_type my_bus_type;

static int my_probe(struct device *dev)
{
    printk("Driver found device which my driver can handle!\n");
    return 0;
}

static int my_remove(struct device *dev)
{
    printk("Driver found device unpluged!\n");
    return 0;
}

struct device_driver my_driver = {
    .name = "my_dev",        /* 注意这里和设备名字一样 */
    .bus = &my_bus_type,     /* 挂接我们实现的总线 */
    .probe = my_probe,
    .remove = my_remove,
};


/*
 * Export a simple attribute.
 */
static ssize_t mydriver_show(struct device_driver *driver, char *buf)
{
    return sprintf(buf, "%s\n", "This is my driver!");
}

/* 定义了bus_attr_drv属性 */
static DRIVER_ATTR(drv, S_IRUGO, mydriver_show, NULL);

static int __init my_driver_init(void)
{
    int ret = 0;

    /*注册驱动*/
    ret = driver_register(&my_driver);

    /*创建属性文件*/
    ret = driver_create_file(&my_driver, &driver_attr_drv);

    return ret;

}

static void my_driver_exit(void)
{
    driver_unregister(&my_driver);
}

module_init(my_driver_init);
module_exit(my_driver_exit);
```



```
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus/yeshen_bus/drivers
root@socfpga_cyclone5:/lib/modules/3.7.0# insmod driver.ko
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus/yeshen_bus/drivers
my_dev
root@socfpga_cyclone5:/lib/modules/3.7.0#
```

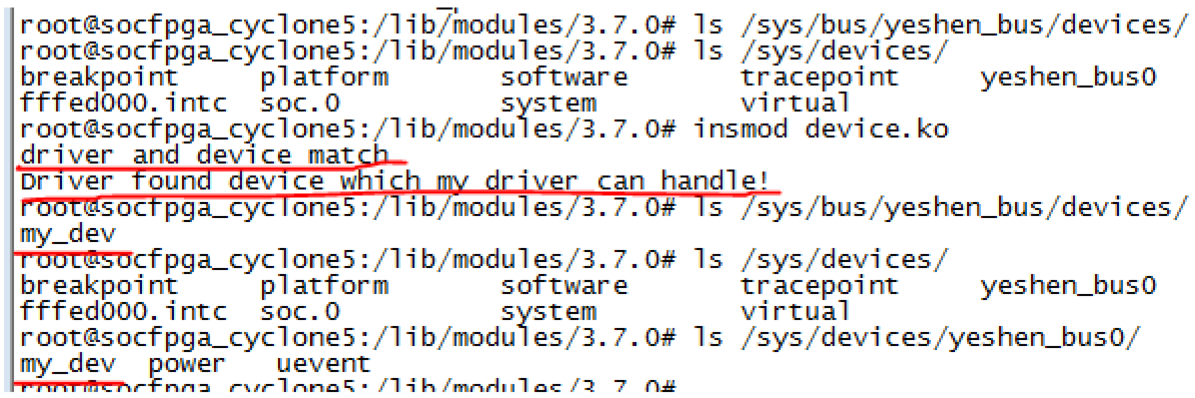
在insmod驱动模块后驱动挂接到了对应总线的drivers目录。

3.实现一个挂接在上面总线的设备

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

```c
#include <linux/string.h>


MODULE_LICENSE("Dual BSD/GPL");

/* 使用bus模块中导出的符号 */
extern struct device my_bus;
extern struct bus_type my_bus_type;


/* Why need this ?*/
static void my_dev_release(struct device *dev)
{

}

struct device my_dev = {
    .bus = &my_bus_type,
    .parent = &my_bus,
    .release = my_dev_release,
};


/*
 * Export a simple attribute.
 */
static ssize_t mydev_show(struct device *dev, struct device_attribute *attr,
            char *buf)
{
    return sprintf(buf, "%s\n", "This is my device!");
}

/* 定义了bus_attr_dev属性 */
static DEVICE_ATTR(dev, S_IRUGO, mydev_show, NULL);

static int __init my_device_init(void)
{
    int ret = 0;

    /* 初始化设备 */
    dev_set_name(&my_dev, "my_dev");

    /*注册设备*/
    ret = device_register(&my_dev);

    /*创建属性文件*/
    device_create_file(&my_dev, &dev_attr_dev);

    return ret;

}
```

```
static void my_device_exit(void)
{
    device_unregister(&my_dev);
}

module_init(my_device_init);
module_exit(my_device_exit);
```



```
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus/yeshen_bus/devices/
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/devices/
breakpoint     platform       software       tracepoint     yeshen_bus0
fffed000.intc  soc.0          system         virtual
root@socfpga_cyclone5:/lib/modules/3.7.0# insmod device.ko
driver and device match
Driver found device which my driver can handle!
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/bus/yeshen_bus/devices/
my_dev
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/devices/
breakpoint     platform       software       tracepoint     yeshen_bus0
fffed000.intc  soc.0          system         virtual
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /sys/devices/yeshen_bus0/
my_dev  power   uevent
root@socfpga_cyclone5:/lib/modules/3.7.0#
```

在insmod设备后总线的my_match匹配驱动和设备成功，打印driver and device match，然后调用驱动的my_probe打印Driver found device which my driver can handle!。设备挂接到了对应总线的devices目录。

然后还要注意/sys/devices中没有my_dev，而是出现到了/sys/devices/yeshen_bus0中，因为my_dev结构体中的parent赋了my_bus的地址，而不是空指针。