PCI device driver and virtual network card driver source code analysis

转载 Kernel base camp Posted at 2022-08-24 19:53:11 155 Favorite 2 copyright

Category column: Linux kernel C/C++ Article tags: network linux c++ interview server

Linux kernel Included by 2 columns at the same time 0 subscribe 44 articles subscribe column

Virtual network card driver routine

```
#include<linux/module.h>
#include<linux/sched.h>
#include<linux/kernel.h>
#include<linux/slab.h>
#include<linux/errno.h>
#include<linux/types.h>
#include<linux/interrupt.h>
#include<linux/in.h>
#include<linux/netdevice.h>
#include<linux/etherdevice.h>
#include<linux/ip.h>
#include<linux/tcp.h>
#include<linux/skbuff.h>
#include<linux/if_ether.h>
#include<linux/in6.h>
#include<asm/uaccess.h>
#include<asm/checksum.h>
#include<linux/platform_device.h>

#define  MAC_AUTO
static struct net_device *vir_net_devs;

struct vir_net_priv {
    struct net_device_stats stats;      //有用的统计信息
    int status;                         //网络设备的状态信息，是发完数据包，还是接收到网络数据包
    int rx_packetlen;                   //接收到的数据包长度
    u8 *rx_packetdata;                  //接收到的数据
    int tx_packetlen;                   //发送的数据包长度
    u8 *tx_packetdata;                  //发送的数据
    struct sk_buff *skb;                //socket buffer结构体，网络各层之间传送数据都是通过这个结构体来实现的
    spinlock_t lock;                    //自旋锁
};

/*网络设备开启时会执行该函数*/
int vir_net_open(struct net_device *dev) {
#ifndef MAC_AUTO
    int i;
    for (i=0; i<6; i++) {
        dev->dev_addr[i] = 0xaa;
    }
#else
    random_ether_addr(dev->dev_addr);
#endif
    /*打开传输队列进行数据传输*/
    netif_start_queue(dev);
    printk("vir_net_open\n");
    return 0;
}

/*关闭的时候，关闭队列*/
```

```
int vir_net_release(struct net_device *dev) {          /*停止发送数据*/
    netif_stop_queue(dev);
    printk("vir_net_release\n");
    return 0;
}

/*接包函数,有数据过来时, 中断执行*/
void vir_net_rx(struct net_device *pdev, int len, unsigned char *buf) {
    struct sk_buff *skb;
    struct vir_net_priv *priv = (struct vir_net_priv *) pdev->ml_priv;
    skb = dev_alloc_skb(len+2);//分配一个socket buffer,并且初始化skb->data,skb->tail和skb->head
    if(!skb) {
        printk("gecnet rx: low on mem - packet dropped\n");
        priv->stats.rx_dropped++;
        return;
    }
    skb_reserve(skb, 2); /* align IP on 16B boundary */
    memcpy(skb_put(skb, len), buf, len);//skb_put是把数据写入到socket buffer
    /* Write metadata, and then pass to the receive level */
    skb->dev = pdev;
    skb->protocol = eth_type_trans(skb, pdev);//返回的是协议号
    skb->ip_summed = CHECKSUM_UNNECESSARY; //此处不校验
    priv->stats.rx_packets++;//接收到包的个数+1

    priv->stats.rx_bytes += len;//接收到包的长度
    printk("vir_net_rx\n");
    netif_rx(skb);//通知内核已经接收到包, 并且封装成socket buffer传到上层
    return;
}

/*模拟硬件发送数据*/
void vir_net_hw_tx(char *buf, int len, struct net_device *dev) {
    struct net_device *dest;//目标设备结构体, net_device存储一个网络接口的重要信息, 是网络驱动程序的核心
    struct vir_net_priv *priv;

    if (len < sizeof(struct ethhdr) + sizeof(struct iphdr)) {
        printk("vir_net: packet too short (%i octets)\n", len);
        return;
    }
    dest = vir_net_devs;
    priv = (struct vir_net_priv *)dest->ml_priv;
    priv->rx_packetlen = len;
    priv->rx_packetdata = buf;

    printk("vir_net_hw_tx\n");
    dev_kfree_skb(priv->skb);
}


/*发包函数, 上层有数据发送时, 该函数会被调用*/
int vir_net_tx(struct sk_buff *skb, struct net_device *pdev) {
    int len;
    char *data;
    struct vir_net_priv *priv = (struct vir_net_priv *)pdev->ml_priv;
    if(skb == NULL) {
        printk("net_device = %p, skb = %p\n", pdev, skb);
        return 0;
    }
    /*ETH_ZLEN是所发的最小数据包的长度*/
    len = skb->len < ETH_ZLEN ? ETH_ZLEN : skb->len;
    /*将要发送的数据包中数据部分*/
    data = skb->data;
    priv->skb = skb;
    /*调用硬件接口进行数据的发送*/
    vir_net_hw_tx(data, len, pdev);
    printk("vir_net_tx, pdev = %p\n", pdev);
    return 0;
}

/*设备初始化函数*/
int vir_net_device_init(struct net_device *pdev) {
    /*填充一些以太网中的设备结构体的项*/
    ether_setup(pdev);
    /*keep the default flags, just add NOARP */
    pdev->flags |= IFF_NOARP;
    /*为priv分配内存*/
    pdev->ml_priv = kmalloc(sizeof(struct vir_net_priv), GFP_KERNEL);
    if (pdev->ml_priv == NULL){
        return -ENOMEM;
    }
    memset(pdev->ml_priv, 0, sizeof(struct vir_net_priv));
    spin_lock_init(&((struct vir_net_priv *)pdev->ml_priv)->lock);
    printk("vir_net_device_init, pdev = %p\n", pdev);
    return 0;
}


/*结构体填充*/
static const struct net_device_ops vir_net_netdev_ops = {
    .ndo_open       = vir_net_open,       //打开网卡 对应ifconfig xx up
    .ndo_stop       = vir_net_release,    //关闭网卡 对应ifconfig xx down
    .ndo_start_xmit = vir_net_tx,         //开启数据包传输(对应上层要发送数据时)
    .ndo_init       = vir_net_device_init,       //初始化网卡硬件
};


/**/
static void vir_plat_net_release(struct device *pdev) {
    printk("vir_plat_net_release, pdev = %p\n", pdev);
}


/*匹配*/
static int vir_net_probe(struct platform_device *pdev) {
    int result = 0;
    /*vir_net_devs结构体相当于一个虚拟的网络设备*/
    vir_net_devs = alloc_etherdev(sizeof(struct net_device));
    vir_net_devs->netdev_ops = &vir_net_netdev_ops;
    strcpy(vir_net_devs->name, "net_0");
    /*上面填充了3项，如果是真实的网卡会填充更多，然后
    使用register_netdev进行注册，net/core,注册好了以后
    内核当中就会有这个设备了，当这个网络设备up以后就会进入open函数*/
    if ((result = register_netdev(vir_net_devs))) {
        printk("vir_net: error %i registering device \"%s\"\n", result, vir_net_devs->name);
    }
    printk("vir_net_probe, pdev = %p\n", pdev);
    return 0;
}

/*设备移除函数*/
static int  vir_net_remove(struct platform_device *pdev) {
    kfree(vir_net_devs->ml_priv);
    unregister_netdev(vir_net_devs);
    return 0;
}

/*结构体填充*/
static struct platform_device vir_net= {
    .name = "vir_net",
                           .id   = -1,
    .dev  = {
    .release = vir_plat_net_release,
    },
};

/*结构体填充*/
static struct platform_driver vir_net_driver = {
    .probe  = vir_net_probe,
    .remove  = vir_net_remove,
    .driver  = {
    .name = "vir_net",    /*这里的name和上面那个结构体的name如果匹配就会执行probe函数*/
    .owner = THIS_MODULE,
    },
};

/*模块入口函数*/
static int __init vir_net_init(void) {
    printk("vir_net_init\n");
    platform_device_register(&vir_net);
    return platform_driver_register(&vir_net_driver);
}

/*模块退出函数*/
static void __exit vir_net_exit(void) {
    printk("vir_net_exit\n");
    platform_driver_unregister(&vir_net_driver);
    platform_device_unregister(&vir_net);
}

module_init(vir_net_init);
module_exit(vir_net_exit);
MODULE_LICENSE("GPL");
```

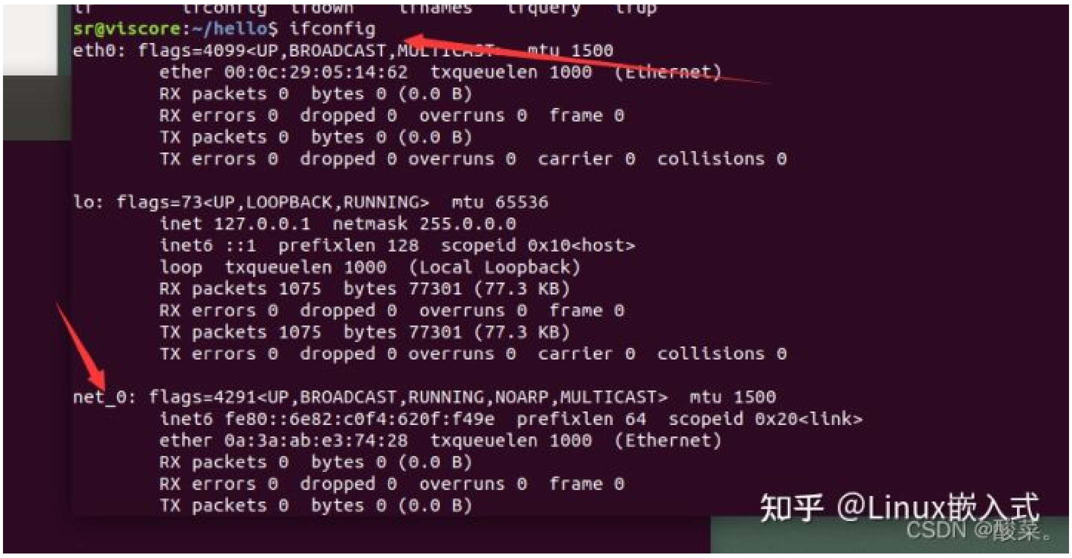
第0000讲 Linux内核源码分析课程介绍 V4.0
第000a讲 Linux内核整体架构与学习路线（补充）-OK
第000b讲 实战操作：编译自己Linux内核（补充）-OK
第001讲 Linux内核源码《目录组织结构详解》-OK
第002讲 进程原理及系统调用-OK
第003讲 调度器及CFS调度器-OK
第004讲 1实时调度类及SMP和NUMA-OK
第004讲 实战操作-2进程优先级与调度策略-OK
第005讲 RCU机制及内存优化屏障-OK
第006讲 内核内存布局和堆管理-OK
第007讲 虚拟地址空间布局架构
第008讲 内存映射原理

第025讲 通用文件模型及VFS结构
第026讲 处理VFS对象及标准函数
第027讲 1Ext文件系统族-Ext2文件系统
第027讲 2Ext文件系统族-Ext4_日志JBD2
第028讲 proc文件系统
第029讲 简单文件系统
第030讲 挂载文件系统
第031讲 套接字及分层模型
第032讲 套接字缓冲区及Net_device
第033讲 Linux内核邻接子系统分析
第034讲 内核Netlink套接字
第035讲 网络层分析

Makefile

```
CONFIG_MODULE_SIG = n
obj-m+=virnet.o
all:
        make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

When using sudo ifconfig xxx up/down and sudo insmod xxx, sudo rmmod xxx commands, the output results can be observed by opening another terminal, tail -f /var/log/kern.log, and viewing the output of the kernel log to observe When it comes to the execution of each command, which function will they execute.



```
if          ifconfig   ifdown     ifnames    ifquery    ifup
sr@viscore:~/hello$ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:0c:29:05:14:62  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1075  bytes 77301 (77.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1075  bytes 77301 (77.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

net_0: flags=4291<UP,BROADCAST,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet6 fe80::6e82:c0f4:620f:f49e  prefixlen 64  scopeid 0x20<link>
        ether 0a:3a:ab:e3:74:28  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
```

PCI device driver routine

```
/*必须包含的两个头文件*/
#include <linux/module.h>
#include <linux/pci.h>

/*自定义结构体，用在中断服务函数里面*/
struct pci_Card {
   resource_size_t io;      /*端口读写变量*/
   long range, flags;       /*io地址范围，标志位*/
   void __iomem *ioaddr;    /*io映射地址*/
   int irq;                 /*中断号*/
};

/*驱动程序支持的设备列表，如果有匹配的设备，那么改驱动程序就会被执行*/
static struct pci_device_id ids[] = {
    {
        PCI_DEVICE(PCI_VENDOR_ID_INTEL, /*vendor id, 厂家ID*/
        0x100f)                         /*device id, 设备ID，可以通过lspci或者去相应的目录里面看*/
    },

    {
        PCI_DEVICE(PCI_VENDOR_ID_INTEL,
        PCI_DEVICE_ID_INTEL_80332_0)
    },

    {0,} /*最后一组为0，表示结束*/
};

/*进行注册,pci总线，ids为上面定义的设置*/
MODULE_DEVICE_TABLE(pci, ids);

/*打印配置空间里面的一些信息*/
void skel_get_configs(struct pci_dev *dev) {

    uint8_t revisionId;
    uint16_t vendorId, deviceId;
    uint32_t classId;

    /*
    从参数dev里面也是可以打印vendorID等信息的，
    这个结构体里面包含这个成员变量，用下面的API     获取得到的结果也是一致的
    */

    pci_read_config_word(dev, PCI_VENDOR_ID, &vendorId);
    printk("vendorID = %x", vendorId);

    pci_read_config_word(dev, PCI_DEVICE_ID, &deviceId);
    printk("deviceID = %x", deviceId);

    pci_read_config_byte(dev, PCI_REVISION_ID, &revisionId);
    printk("revisionID = %x",revisionId);

    pci_read_config_dword(dev, PCI_CLASS_REVISION, &classId);
    printk("classID = %x",classId);
}

/*设备中断服务函数*/
static irqreturn_t pci_Mcard_interrupt(int irq, void *dev_id) {
   struct pci_Card *pci_Mcard = (struct pci_Card *)dev_id;
   /*中断函数里面打印中断号*/
   printk("irq = %d, pci_Mcard_irq = %d\n", irq, pci_Mcard->irq);
   return IRQ_HANDLED;
}

/*有匹配的设备，这个函数会执行*/
static int probe(struct pci_dev *dev, const struct pci_device_id *id) {
    int retval = 0;
    struct pci_Card *pci_Mcard;
    printk("probe func\n");

    /*设备使能*/
    if(pci_enable_device(dev)) {
        printk (KERN_ERR "IO Error.\n");
        return -EIO;
    }

    pci_Mcard = kmalloc(sizeof(struct pci_Card),GFP_KERNEL);
    if(!pci_Mcard) {
        printk("In %s,kmalloc err!",__func__);
        return -ENOMEM;
    }

    /*设备中断号*/
    pci_Mcard->irq = dev->irq;
    if(pci_Mcard->irq < 0) {
        printk("IRQ is %d, it's invalid!\n",pci_Mcard->irq);
        goto out_pci_Mcard;
    }

    /*获取io内存相关信息*/
    pci_Mcard->io = pci_resource_start(dev, 0);
    pci_Mcard->range = pci_resource_end(dev, 0) - pci_Mcard->io + 1;
    pci_Mcard->flags = pci_resource_flags(dev,0);
    printk("start %llx %lx %lx\n",pci_Mcard->io, pci_Mcard->range, pci_Mcard->flags);
    printk("PCI base addr 0 is io%s.\n",(pci_Mcard->flags & IORESOURCE_MEM)? "mem":"port");

    /*防止地址访问冲突，所以这里先申请*/
    retval = pci_request_regions(dev,"pci_module");
    if(retval) {
        printk("PCI request regions err!\n");
        goto out_pci_Mcard;
    }

    /*再进行映射*/
    pci_Mcard->ioaddr = pci_ioremap_bar(dev, 0);
    if(!pci_Mcard->ioaddr) {
      printk("ioremap err!\n");
      retval = -ENOMEM;
      goto out_regions;
    }

    /*申请中断IRQ并设定中断服务子函数*/
    retval = request_irq(pci_Mcard->irq, pci_Mcard_interrupt, IRQF_SHARED, "pci_module", pci_Mcard);
    if(retval) {
      printk (KERN_ERR "Can't get assigned IRQ %d.\n",pci_Mcard->irq);
      goto out_iounmap;
    }

    pci_set_drvdata(dev, pci_Mcard);
    skel_get_configs(dev);
    return 0;

out_iounmap:
    iounmap(pci_Mcard->ioaddr);
out_regions:
    pci_release_regions(dev);
out_pci_Mcard:
    kfree(pci_Mcard);
    return retval;
}

/*移除PCI设备*/
static void remove(struct pci_dev *dev) {
   struct pci_Card *pci_Mcard = pci_get_drvdata(dev);
   free_irq (pci_Mcard->irq, pci_Mcard);
   iounmap(pci_Mcard->ioaddr);
   pci_release_regions(dev);
   kfree(pci_Mcard);
   pci_disable_device(dev);
   printk("remove pci device ok\n");
}

/*结构体成员变量填充*/
static struct pci_driver pci_driver = {
    .name = "pci_module",
    .id_table = ids,
    .probe = probe,
    .remove = remove,
};

/*模块入口函数*/
static int __init pci_module_init(void) {
    printk("pci module entry function\n");
    return pci_register_driver(&pci_driver);
}

/*模块退出函数*/
static void __exit pci_module_exit(void) {
    printk("pci module exit function\n");
    pci_unregister_driver(&pci_driver);
}

MODULE_LICENSE("GPL");

module_init(pci_module_init);
module_exit(pci_module_exit);
```

Makefile

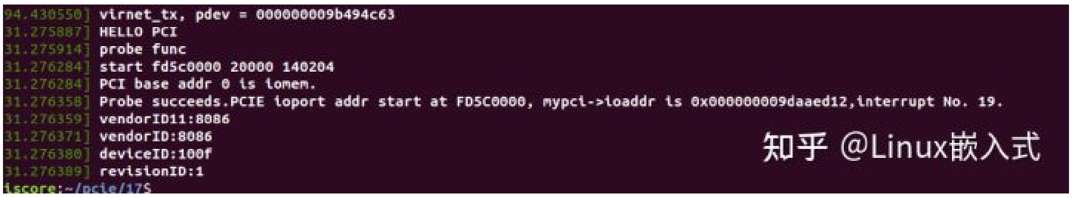
```
obj-m   := pci_module.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD       := $(shell pwd)

all:
        $(MAKE) -C $(KERNELDIR) M=$(PWD)
clean:
```

Uninstall e1000 before execution

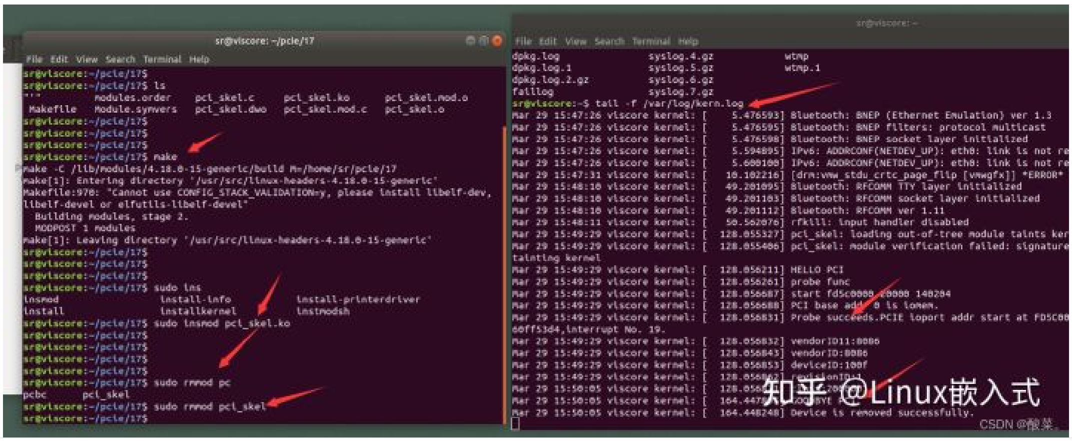
```
sudo rmmod e1000
```

Then, execute

```
sudo insmod pci_module.ko
dmesg
```



```
94.430550] virnet_tx, pdev = 000000009b494c63
31.275887] HELLO PCI
31.275914] probe func
31.276284] start fd5c0000 20000 140204
31.276284] PCI base addr 0 is iomem.
31.276358] Probe succeeds.PCIE ioport addr start at FD5C0000, mypci->ioaddr is 0x000000009daaed12,interrupt No. 19.
31.276359] vendorID11:8086
31.276371] vendorID:8086
31.276380] deviceID:100f
31.276389] revisionID:1
iscore:~/pcie/17$
```



```
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$ ls
'''           modules.order    pci_skel.c     pci_skel.ko      pci_skel.mod.o
 Makefile     Module.symvers   pci_skel.dwo   pci_skel.mod.c   pci_skel.o
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$ make
make -C /lib/modules/4.18.0-15-generic/build M=/home/sr/pcie/17
make[1]: Entering directory '/usr/src/linux-headers-4.18.0-15-generic'
Makefile:970: "Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev,
libelf-devel or elfutils-libelf-devel"
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.18.0-15-generic'
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$ sudo ins
insmod                 install-info           install-printerdriver
install                installkernel          instmodsh
sr@viscore:~/pcie/17$ sudo insmod pci_skel.ko
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$
sr@viscore:~/pcie/17$ sudo rmmod pc
pcbc      pci_skel
sr@viscore:~/pcie/17$ sudo rmmod pci_skel
sr@viscore:~/pcie/17$
```

```
dpkg.log                syslog.4.gz            wtmp
dpkg.log.1              syslog.5.gz            wtmp.1
dpkg.log.2.gz           syslog.6.gz
faillog                 syslog.7.gz
sr@viscore:~$ tail -f /var/log/kern.log
Mar 29 15:47:26 viscore kernel: [    5.476593] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
Mar 29 15:47:26 viscore kernel: [    5.476595] Bluetooth: BNEP filters: protocol multicast
Mar 29 15:47:26 viscore kernel: [    5.476598] Bluetooth: BNEP socket layer initialized
Mar 29 15:47:26 viscore kernel: [    5.594895] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not re
Mar 29 15:47:26 viscore kernel: [    5.600100] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not re
Mar 29 15:47:31 viscore kernel: [   10.102216] [drm:vmw_stdu_crtc_page_flip [vmwgfx]] *ERROR*
Mar 29 15:48:10 viscore kernel: [   49.201095] Bluetooth: RFCOMM TTY layer initialized
Mar 29 15:48:10 viscore kernel: [   49.201103] Bluetooth: RFCOMM socket layer initialized
Mar 29 15:48:10 viscore kernel: [   49.201112] Bluetooth: RFCOMM ver 1.11
Mar 29 15:48:11 viscore kernel: [   50.562076] rfkill: input handler disabled
Mar 29 15:49:29 viscore kernel: [  128.055327] pci_skel: loading out-of-tree module taints ker
Mar 29 15:49:29 viscore kernel: [  128.055406] pci_skel: module verification failed: signature
tainting kernel
Mar 29 15:49:29 viscore kernel: [  128.056211] HELLO PCI
Mar 29 15:49:29 viscore kernel: [  128.056261] probe func
Mar 29 15:49:29 viscore kernel: [  128.056687] start fd5c0000 20000 140204
Mar 29 15:49:29 viscore kernel: [  128.056688] PCI base addr 0 is iomem.
Mar 29 15:49:29 viscore kernel: [  128.056831] Probe succeeds.PCIE ioport addr start at FD5C00
60ff53d4,interrupt No. 19.
Mar 29 15:49:29 viscore kernel: [  128.056832] vendorID11:8086
Mar 29 15:49:29 viscore kernel: [  128.056843] vendorID:8086
Mar 29 15:49:29 viscore kernel: [  128.056853] deviceID:100f
Mar 29 15:49:29 viscore kernel: [  128.056862] revisionID:1
Mar 29 15:50:05 viscore kernel: [  128.056[illegible]
Mar 29 15:50:05 viscore kernel: [  164.447[illegible] GOODBYE
Mar 29 15:50:05 viscore kernel: [  164.448248] Device is removed successfully.
```

The knowledge points of the article are matched with the official knowledge files, and relevant knowledge can be further learned