PCI bus driver code combing (1) -- the overall framework

原创 Santiago Posted at 2021-01-16 19:30:04 908 Favorite 16 copyright
Category column: Linux Article tags: linux

PCI bus driver code combing (1) – the overall framework

1 Overview

The driver code structure of the PCI bus is slightly different from that of conventional buses such as SPI, IIC, and platform. The first difference is that the code framework of the PCI bus has multiple entry functions. Buses like SPI often only use one or two initcalls. , but the driver framework of the PCI bus uses many different levels of initcall function entries, and we will sort them out in order below.

Note: The code analyzed in this document uses x86 as the hardware platform and uses the ACPI mechanism, so it is slightly different from the PCI bus driver implemented by other platforms such as powerPC. The kernel version analyzed in this document is Linux-4.4.185

2. PCI-related entry functions

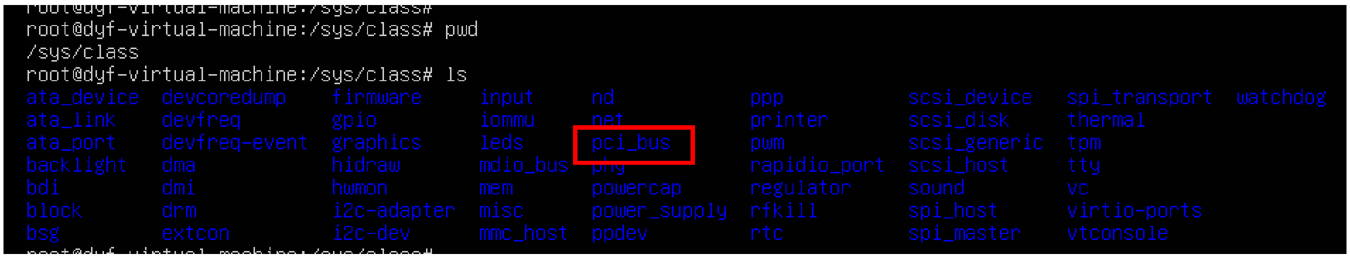
When we compile the Linux kernel source code, we can get a System.map file, which sorts out the order in which the driver calls the initcall macro. Through this file, we can find the first entry function related to the PCI bus.

2.1 pcibus_class_init

Location: /drivers/pci/probe.c

```
static struct class pcibus_class = {
        .name           = "pci_bus",
        .dev_release    = &release_pcibus_dev,  //资源
        .dev_groups     = pcibus_groups,
};

static int __init pcibus_class_init(void)
{
        return class_register(&pcibus_class);
}
postcore_initcall(pcibus_class_init);
```

Function analysis: The execution of this function is also very simple, that is, a class_register function is called to register a pcibus_class structure, and the execution level is 2 (a total of 7 levels). The function of this function is to register a PCI_BUS class, and in / A pci_bus directory is generated under the sys/class directory as shown in the figure below.



```
root@dyf-virtual-machine:/sys/class#
root@dyf-virtual-machine:/sys/class# pwd
/sys/class
root@dyf-virtual-machine:/sys/class# ls
ata_device  devcoredump    firmware     input     nd            ppp           scsi_device   spi_transport  watchdog
ata_link    devfreq        gpio         iommu     net           printer       scsi_disk     thermal
ata_port    devfreq-event  graphics     leds      pci_bus       pwm           scsi_generic  tpm
backlight   dma            hidraw       mdio_bus  pnp           rapidio_port  scsi_host     tty
bdi         dmi            hwmon        mem       powercap      regulator     sound         vc
block       drm            i2c-adapter  misc      power_supply  rfkill        spi_host      virtio-ports
bsg         extcon         i2c-dev      mmc_host  ppdev         rtc           spi_master    vtconsole
root@dyf-virtual-machine:/sys/class#
```

2.2 pci_driver_init

Location: /drivers/pci/pci-driver.c

```
struct bus_type pci_bus_type = {
        .name           = "pci",
        .match          = pci_bus_match,
```

Santiago focus on

```
        .uevent         = pci_uevent,         //用户事件
        .probe          = pci_device_probe,   //匹配后的执行函数
        .remove         = pci_device_remove,  //退出函数
        .shutdown       = pci_device_shutdown,
        .dev_groups     = pci_dev_groups,
        .bus_groups     = pci_bus_groups,
        .drv_groups     = pci_drv_groups,
        .pm             = PCI_PM_OPS_PTR,         //电源管理相关
};
EXPORT_SYMBOL(pci_bus_type);

static int __init pci_driver_init(void)
{
        return bus_register(&pci_bus_type);
}
postcore_initcall(pci_driver_init);
```

Function analysis: This is the second startup function related to the PCI bus. This function is similar to pcibus_class_init, and only completes the registration of a structure. The startup level is 2. It is used to register a pci directory under /sys/bus and Complete the creation of subdirectories under this directory (device directory and driver directory)

2.3 pci_arch_init

Location: /arch/x86/pci/init.c

```
static __init int pci_arch_init(void)
{
#ifdef CONFIG_PCI_DIRECT      //CONFIG_PCI_DIRECT此选项打开
        int type = 0;

        type = pci_direct_probe();      //config1方法配置
#endif

        if (!(pci_probe & PCI_PROBE_NOEARLY))
                pci_mmcfg_early_init();   //MMconfig方法配置

        if (x86_init.pci.arch_init && !x86_init.pci.arch_init())
                return 0;

#ifdef CONFIG_PCI_BIOS           //未执行
        pci_pcbios_init();
#endif

#ifdef CONFIG_PCI_DIRECT        //
        pci_direct_init(type);
#endif
    ......
        return 0;
}
arch_initcall(pci_arch_init);
```

Function analysis: The function internally uses conditional compilation to determine the execution of internal functions. We can get all the macro definitions defined in the source code through the .config file. After checking, it is found that CONFIG_PCI_DIRECT is defined and CONFIG_PCI_BIOS is not defined. The function of this function is to set the entire PCI configuration. How to read and write spaces.

2.4 pci_slot_init

Location: /drivers/pci/slot.c

```

static int pci_slot_init(void)
{
        struct kset *pci_bus_kset;

        pci_bus_kset = bus_get_kset(&pc
```

Santiago focus on

```
        pci_slots_kset = kset_create_and_add("slots", NULL,
                                                &pci_bus_kset->kobj);                //创建一个slots目录
        if (!pci_slots_kset) {
                printk(KERN_ERR "PCI: Slot initialization failure\n");
                return -ENOMEM;
        }
        return 0;
}
```

Function analysis: The function of this function is to create a subdirectory slot in the /sys/bus/pci directory, indicating the slot. This directory mainly stores the PCI/PCIE slot nodes on the current hardware board. If there is no outgoing PCI/PCIE slot, then this folder is empty. In this function, the main core calling function is kset_create_and_add, which creates a folder named slots under the pci bus.

2.5 acpi_init

Location: /drivers/acpi/bus.c

```
static int __init acpi_init(void)
{
        int result;

    。。。。。。

        pci_mmcfg_late_init();  //pci_mmcfg的第二次配置
        acpi_scan_init();       //acpi设备扫描（包括PCI设备）
        acpi_ec_init();
        acpi_debugfs_init();
        acpi_sleep_proc_init();
        acpi_wakeup_device_init();
        return 0;
}
```

Function analysis: In the previous X86 architecture system, the system will call pcibios_scan_root to enumerate the PCI bus, so this function is often encountered when looking up information, but now the ACPI mechanism is popular in x86 architecture systems, so, The implementation of the enumeration part code has also been changed. This function is the initialization function of ACPI, and the startup level is 4. Regardless of pci_mmcfg_late_init() (this function is the second configuration of pci_mmcfg, because we have completed this before. The relevant configuration of the method, so this function returns without performing any operations, unless there is a problem with your previous configuration), the first important task of ACPI initialization work is to detect the device, and the PCI device we are most concerned about The enumeration operation is also done here.

2.6 pci_subsys_init

Location: /arch/x86/pci/legacy.c

```

static int __init pci_subsys_init(void)
{
        if (x86_init.pci.init())
                pci_legacy_init();    //不执行

        pcibios_fixup_peer_bridges();
        x86_init.pci.init_irq();
        pcibios_init();   //调用pcibios_resource_survey

        return 0;
}

void __init pcibios_resource_survey(void)
{
        struct pci_bus *bus;

        DBG("PCI: Allocating resources\r
```

Santiago focus on

```

        list_for_each_entry(bus, &pci_root_buses, node)
                pcibios_allocate_bus_resources(bus);          //为PCI桥分配地址空间

        //为PCI设备分配地址空间（BIOS中以启用的）
        list_for_each_entry(bus, &pci_root_buses, node)
                pcibios_allocate_resources(bus, 0);

        //为PCI设备分配地址空间（其他PCI设备）
        list_for_each_entry(bus, &pci_root_buses, node)
                pcibios_allocate_resources(bus, 1);

        e820_reserve_resources_late();
        ioapic_insert_resources();
}
```

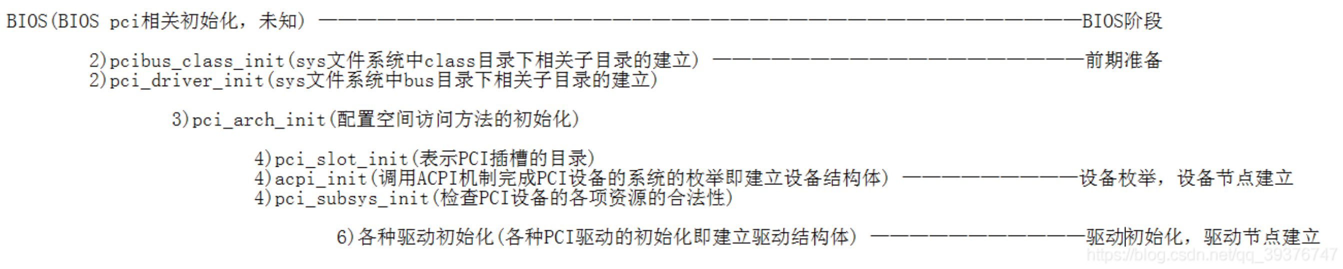
Function analysis: This function mainly calls the pcibios_resource_survey function in pcibios_init to check the legality of resources in these pci_bus structures. And allocate some space from the resource address space of the upper bus for our current PCI device.

3. Summary

When the system is initialized, these functions will be executed sequentially according to the initialization execution level of these initialization functions. The system’s initialization of PCI is roughly divided into four stages:

1.BIOS enumerates PCI devices for the first time
2.Preliminary preparations for PCI device system enumeration (establishment of file system related directories, initialization of access methods)
3.Enumeration of PCI device system stage (establishment of pci_dev)
4. Initialization of PCI device driver (creation of driver)

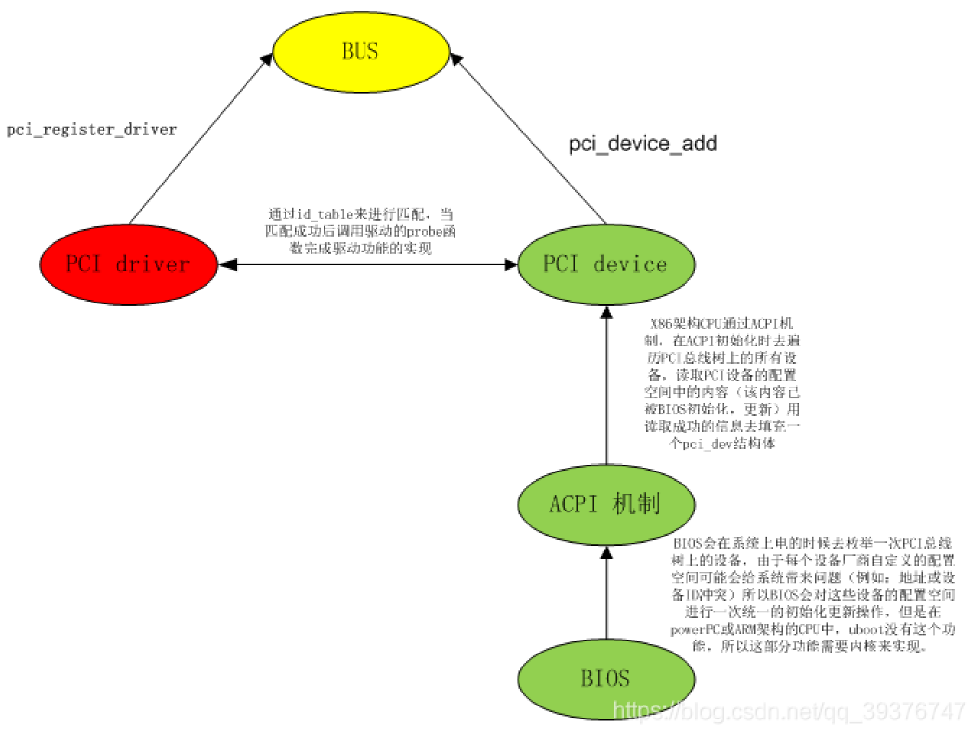
Therefore, the initialization of the PCI bus will have a large number of entry functions. Next, we will make a map of the execution functions of these four stages:



```
BIOS(BIOS pci相关初始化，未知) ————————————————————————————————————————————BIOS阶段

        2)pcibus_class_init(sys文件系统中class目录下相关子目录的建立) ——————————————————————前期准备
        2)pci_driver_init(sys文件系统中bus目录下相关子目录的建立)

                3)pci_arch_init(配置空间访问方法的初始化)

                        4)pci_slot_init(表示PCI插槽的目录)
                        4)acpi_init(调用ACPI机制完成PCI设备的系统的枚举即建立设备结构体) ——————————设备枚举，设备节点建立
                        4)pci_subsys_init(检查PCI设备的各项资源的合法性)

                                6)各种驱动初始化(各种PCI驱动的初始化即建立驱动结构体) ———————————驱动初始化，驱动节点建立
```

the more complicated process is the establishment of the configuration space method and the enumeration of the device. We have described the two stages in two other articles. The detailed execution process will not be listed here. So far, the PCI-related data structures have been established and the relevant initialization has been completed. However, these initialized devices refer to the PCI devices that are connected to each other. Devices, that is, devices that are connected to the PCI bus before the system is powered on. In addition, there are some devices that are plug-and-play devices. They are added to the bus after the system is powered on and initialized. caution.
Conclusion:
For x86 architecture CPUs,
the establishment of pci_dev is generated by the system scan (if the system cannot find the device due to BIOS scan failure, generally you need to build it yourself)
pci_dre is the driver node we need to complete, and implement probe and other methods.
Overall framework:

Santiago focus on



通过id_table来进行匹配，当匹配成功后调用驱动的probe函数完成驱动功能的实现

X86架构CPU通过ACPI机制，在ACPI初始化时去遍历PCI总线树上的所有设备，读取PCI设备的配置空间中的内容（该内容已被BIOS初始化，更新）用读取成功的信息去填充一个pci_dev结构体

BIOS会在系统上电的时候去枚举一次PCI总线树上的设备，由于每个设备厂商自定义的配置空间可能会给系统带来问题（例如：地址或设备ID冲突）所以BIOS会对这些设备的配置空间进行一次统一的初始化更新操作，但是在powerPC或ARM架构的CPU中，uboot没有这个功能，所以这部分功能需要内核来实现。

Santiago focus on