

实验 05 总线_设备_驱动注册流程详解

5.1 本章导读

在 Linux2.6 之后，Linux 设备驱动分为三个实体总线、设备、驱动，平台总线将设备和驱动匹配。在系统注册任意一个驱动的时候，都会寻找对应的设备；当系统注册设备的时候，系统也会寻找对应的驱动进行匹配。

本节实验通过一张框架图，从理论上给大家分析总线设备驱动三者的关系。

5.1.1 工具

5.1.1.1 硬件工具

PC 机一台

5.1.1.2 软件工具

虚拟机 Ubuntu 系统

Linux 源码 “iTop4412_Kernel_3.0_xxx”（在光盘目录 “/Android 源码” 文件夹下,xxx 表示日期）

5.1.2 预备课程

Linux 常用命令视频教程 “01-烧写、编译以及基础知识视频” → “实验 07-Linux 常用命令”

使用手册 3.3.3 小节 Linux 常用 shell 命令

5.1.3 视频资源

本节配套视频为“视频 05_总线_设备_驱动注册流程详解”

5.2 学习目标

本章需要学习以下内容：

了解 Linux 总线的概念

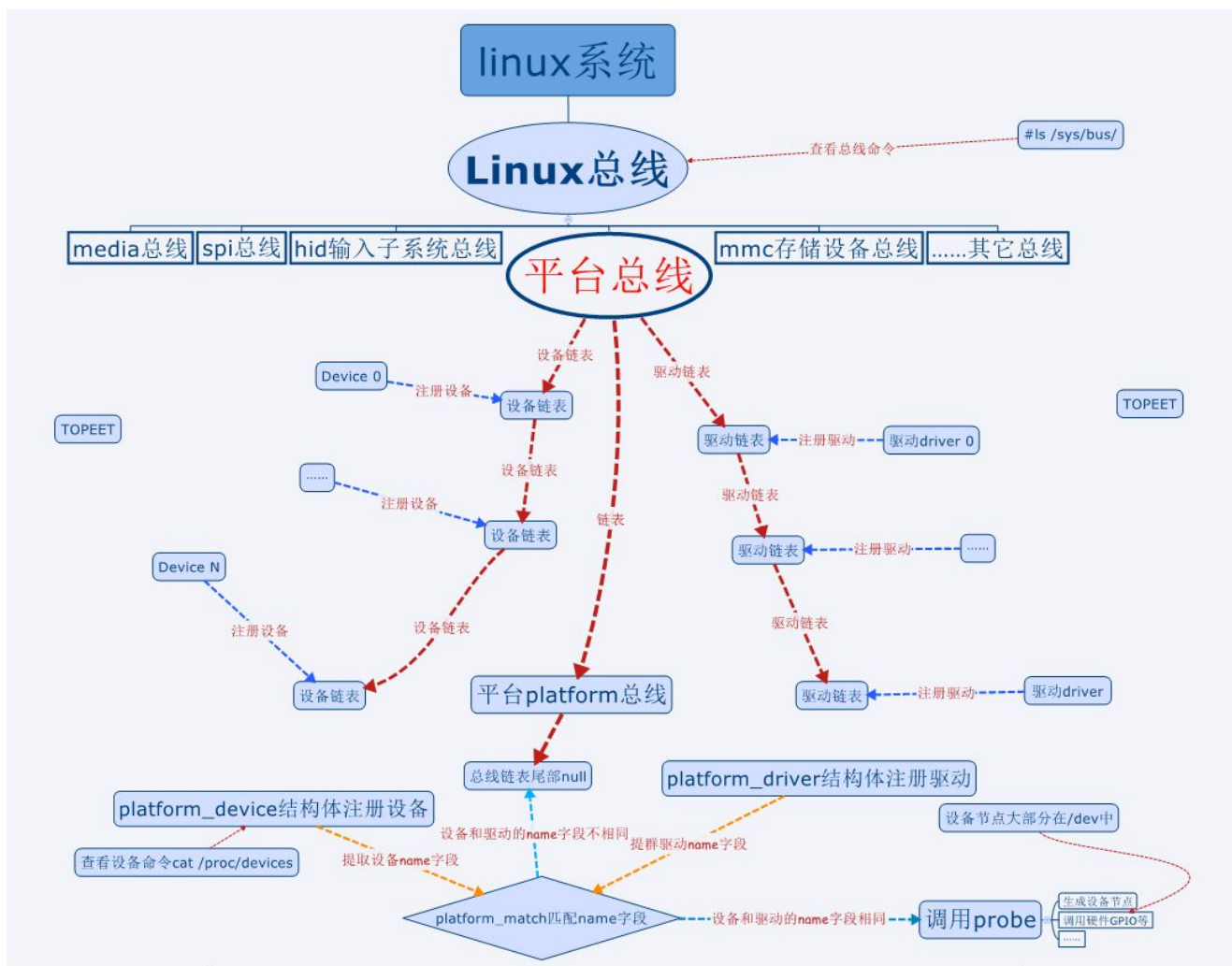
理解平台总线 platform

理解 Linux 设备的概念

掌握 Linux 驱动注册流程

5.3 总线、设备、驱动框架图分析

如下图所示，这是总线、设备、驱动的框架图，下面就来分析它们之间的关系。



5.3.1 总线和平台总线

这里再次强调一个观点，Linux 系统中大部分内容不需要关心，哪怕一些编译进内核的源代码，绝大部分都不需要去阅读。

在 Linux 系统中，任何一个 Linux 设备和 Linux 驱动都是需要挂载到一种总线中。有一些常规的大家容易理解的总线，例如 media 总线、spi 总线、hid 输入子系统总线、eMMC 存储

设备总线等等。假如说设备本身就是一个总线设备，那么挂载到对应的总线上，那是容易理解的。

在任意一个 soc 系统中，都有一些集成的总线，例如在 4412 中就集成了 i2c, spi、usb 等等。针对这些总线设备，它们注册驱动的时候，都会调用对应的总线设备，一个驱动对应一个设备，这个概念很好理解。

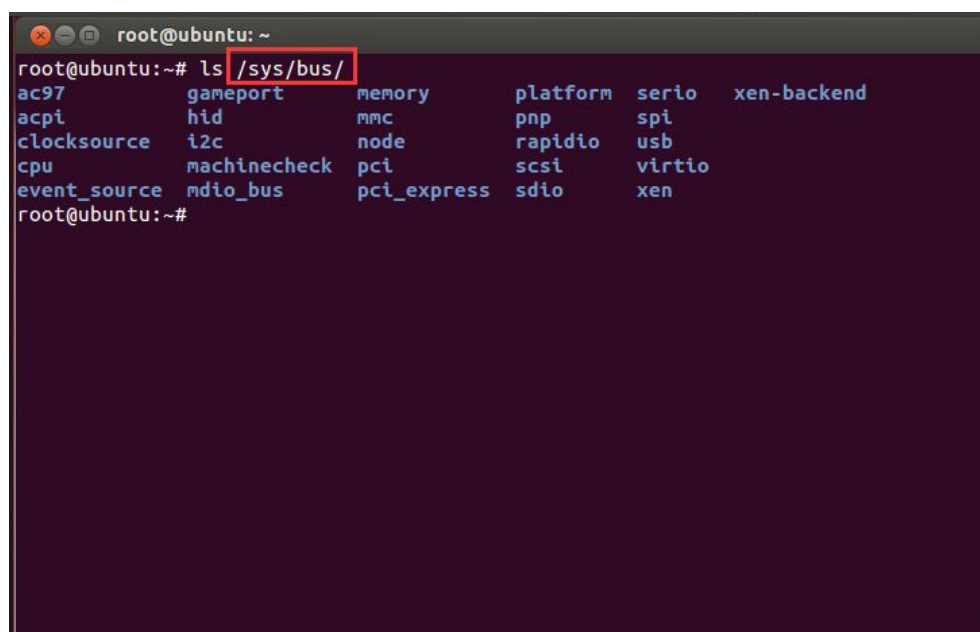
但是还有一些例如 led、蜂鸣器等等一些设备，都不是从字面上理解的总线设备。针对这个情况，Linux 创立了一种虚拟总线，也叫平台总线或者 platform 总线，这个总线也有对应的设备 platform_device,对应的驱动叫 platform_driver。

这里介绍的平台总线，不能够直接和常规的总线对应，只是 Linux 系统提供的一种附加手段，防止 linux 驱动的碎片化，降低 Linux 的使用难度。

另外这里的设备 platform_device 和驱动 platform_driver 也不是和常规的字符设备、块设备、网络设备并列的概念，它只是一种附加的手段。

具体的这个虚拟平台怎么使用，是后续实验要逐渐介绍。

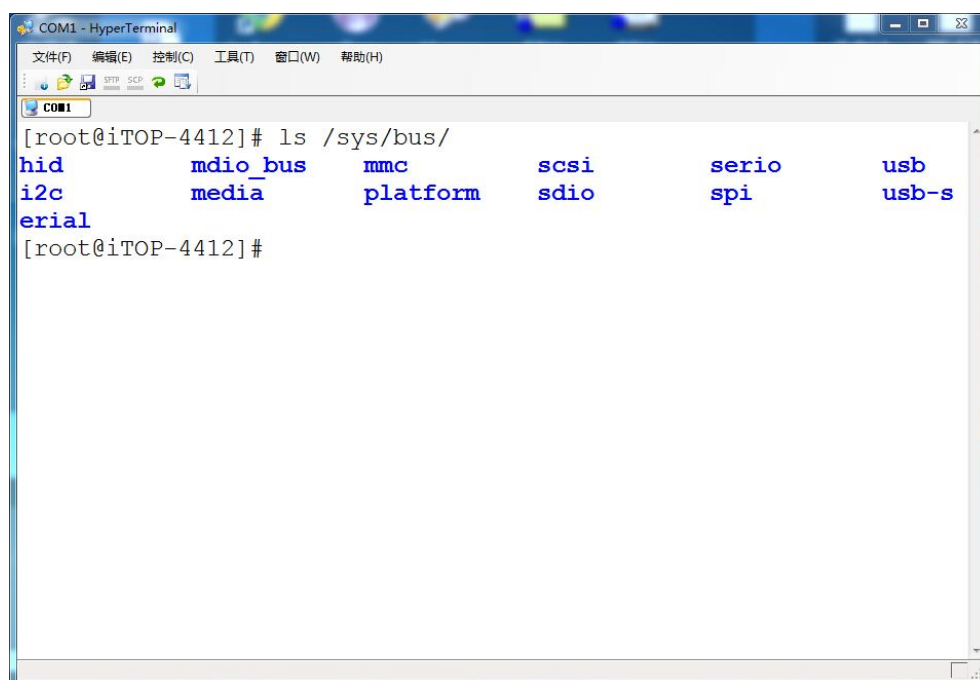
如下图所示，可以通过命令 “ls /sys/bus/” 查看总线，这是在 Ubuntu 系统下使用这个命令，Ubuntu 系统也是属于 Linux 系统，Ubuntu 是属于 Linux 庞大分支中一个拥有图形界面的操作系统。



```
root@ubuntu: ~  
root@ubuntu:~# ls /sys/bus/  
ac97      gameport  memory    platform  serio     xen-backend  
acpi      hid        mmc        pnp        spi  
clocksource i2c        node       rapidio    usb  
cpu       machinecheck pci        scsi       virtio  
event_source mdio_bus  pci_express sdio       xen  
root@ubuntu:~#
```

上图中显示的是 x86 PC 机上的 Ubuntu 总线。

如下图所示，在 iTOP-4412 开发板上运行最小 Linux 系统，使用命令 “ls /sys/bus” ，可以查看开发板带有的总线。其中的 platform 总线是和其它总线在同一个目录下。



```
COM1 - HyperTerminal  
文件(F) 编辑(E) 控制(C) 工具(T) 窗口(W) 帮助(H)  
COM1  
[root@iTOP-4412]# ls /sys/bus/  
hid      mdio_bus  mmc        scsi        serio        usb  
i2c      media     platform   sdio        spi          usb-s  
erial  
[root@iTOP-4412]#
```

5.3.2 Linux 设备

硬件总类繁多,千变万化,一个 USB 接口就可以接无数种键盘、鼠标、存储设备等等。Linux 将设备分为了三大类:字符设备、块设备、网络设备。

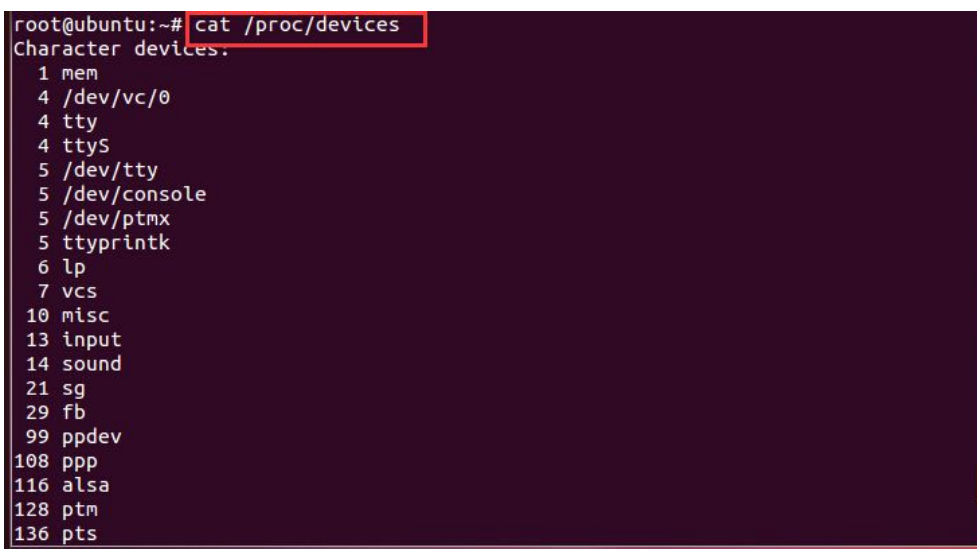
字符设备,字符设备是能够像字节流一样被访问的设备。一般说来对硬件设备 IO 的操作可以归结为字符设备。常见的字符设备有 led、蜂鸣器、串口、键盘等等。

块设备,块设备是通过内存缓冲区访问,可以随机存取的设备,一般性的理解就是存储介质的设备。常见的字符设备有 U 盘、TF 卡、eMMC、电脑硬盘、光盘等等

网络设备,可以和其它主机交换数据的设备。常见的以太网设备、WIFI、蓝牙等。

虽然它们之间有这种官方的分类,但是也没有严格的界限,只是一个比较模糊的划分。

设备种类繁多,不同的设备对应不同的设备名和设备号,在 Ubuntu 虚拟机中使用命令“cat /proc/devices”,如下图所示,可以看到不同的设备都有了编号。



```
root@ubuntu:~# cat /proc/devices
Character devices:
1 mem
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
5 ttyprintk
6 lp
7 vcs
10 misc
13 input
14 sound
21 sg
29 fb
99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
```

启动开发板,在超级终端中输入命令“cat /proc/devices”,可以看到不同的设备。

```
[root@iTOP-4412]# cat /proc/devices
Character devices:
 1 mem
 4 ttyS
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
10 misc
13 input
21 sg
29 fb
81 video4linux
89 i2c
108 ppp
116 alsa
128 ptm
136 pts
150 ...
```

设备号肯定是有限的，一共就只有 256 个主设备号，这里引入了从设备号的概念，理论上就有 256×256 个设备号，这种数量级目前是可以接受的。

5.3.3 Linux 驱动

驱动程序一般以一个模块初始化函数作为入口，例如实验手册“实验二”中最简单的驱动。

在作者看来，驱动的学习分为两大类，一类是需要会写的，主要是字符设备，一类就是移植。

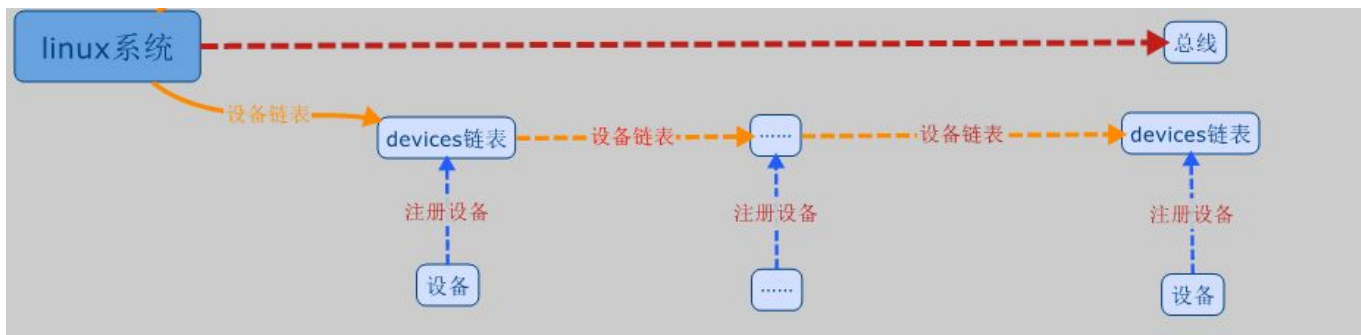
需要会写的就是字符设备，像块设备和网络设备要么是 soc 原厂已经集成，要么是外围器件芯片厂商提供源代码，主要工作在管脚配置和调试。

实验手册主要就是围绕驱动展开的，通过后续的学习大家就可以理解驱动是怎么回事，驱动怎么编写和移植。

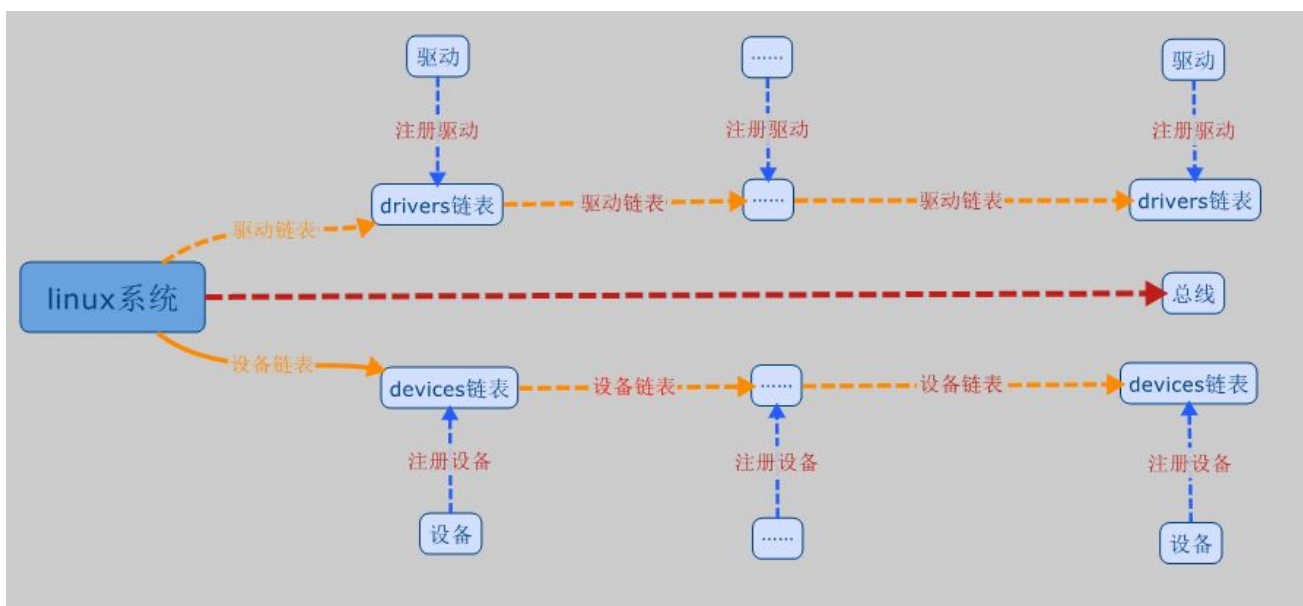
5.3.4 Linux 驱动和设备的注册过程

Linux 内核会要求每出现一个设备就要像总线汇报，或者说注册，出现一个驱动，也要向总线汇报，或者叫注册。

在系统初始化的时候，会扫描连接了哪些设备，并为每一个设备建立一个 struct_device 的变量，然后将设备的变量插入到 devices 链表，如下图所示。



系统初始化任意一个驱动程序的时候，就要准备一个 struct device_driver 结构的变量，然后将驱动的变量插入到 drivers 链表，如下图所示。



Linux 总线是为了将设备和驱动绑定，方便管理。在系统每注册一个设备的时候，会寻找与之匹配的驱动（后面的热拔插设备会用到这个注册流程）；相反，在系统每注册一个驱动的时候，会寻找与之匹配的设备，而匹配由总线完成。

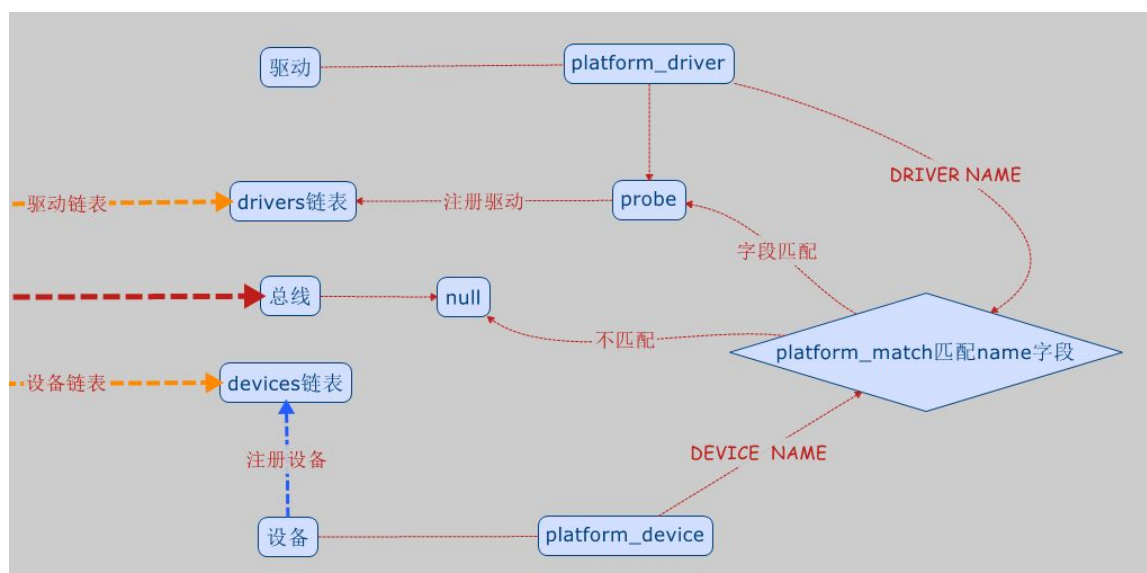
这里只讨论先注册设备再注册驱动的情况。

在注册驱动的时候，系统会通过 platform_match 函数匹配设备和驱动。

注册设备的结构体为 platform_device，注册驱动的结构体为 platform_driver。设备和驱动结构体的成员 name 字段，相同则匹配。

如果匹配了则会调用 platform_driver 中的 probe 函数，注册驱动。

也就是在上图注册驱动的时候要添加一个判断，如下图所示。



大家知道现在有很多设备都是支持热拔插的。在 Linux 中，一般情况都是先注册设备，然后再注册驱动，但是有了热拔插设备之后，情况就不一样了。在热拔插设备中，是有了设备 devices 接入之后，内核会去 driver 链表中寻找寻找驱动。这种情况在后面的实验再讨论。

5.3.5 设备节点简介

在 Linux 系统中，有一个概念叫“一切皆文件”，上层应用使用设备节点访问对应的设备。

设备节点一般是放在“/dev”目录下，在开发板下输入命令“ls /dev”，如下图所示。

```
[root@iTOP-4412]# ls /dev/
AGPS          input          ram0           tty3
HPD           ion            ram1           tty4
adc           keychord       ram10          ttyGS0
alarm         kmem           ram11          ttyGS1
android_adb   kmsg           ram12          ttyGS2
ashmem        leds           ram13          ttyGS3
bus           log            ram14          ttyS0
buzzer_ctl    loop0          ram15          ttyS1
console       loop1          ram2           ttyS2
cpu_dma_latency loop2          ram3           ttyS3
exynos-mem    loop3          ram4           ttySAC0
fb0           loop4          ram5           ttySAC1
fb1           loop5          ram6           ttySAC2
fb10          loop6          ram7           ttySAC3
fb11          loop7          ram8           uinput
fb2           mali           ram9           ump
fb3           mapper         random          urandom
fb4           max485_ctl_pin rc522           usb_accessory
fb5           mem            root            usbdev1.1
fb6           mmcblk0        rtc0            usbdev1.2
fb7           mmcblk0p1      rtc1            usbdev1.3
fb8           mmcblk0p2      s3c-mem         usbdev1.4
fb9           mmcblk0p3      s3c-mfc         video0
fimg2d        mmcblk0p4      sda             video1
full1         mtp_usb        sda1            video11
fuse          network_latency sg0             video12
gps           network_throughput shm             video16
i2c-0         null           snd             video2
i2c-1         pmem           srp             video20
i2c-3         pmem_gpu1     srp_ctrl        video3
i2c-4         ppp            tty             watchdog
i2c-5         ptmx           tty1            xt_qtaguid
i2c-7         pts            tty2            zero
[root@iTOP-4412]#
```

上层应用有一套标准的接口文件和函数用来和底层通信。

Linux 将所有对设备的操作全部都抽象为对文件的操作，常用的文件操作函数有 open、close、write、read、write 等。

后面会详细介绍这些函数如何使用，即使是驱动工程师，也要使用这些函数编写简单的驱动测试函数，这些函数的使用必须掌握的。