《嵌入式系统》

（第九讲）

厦门大学信息学院软件工程系 曾文华

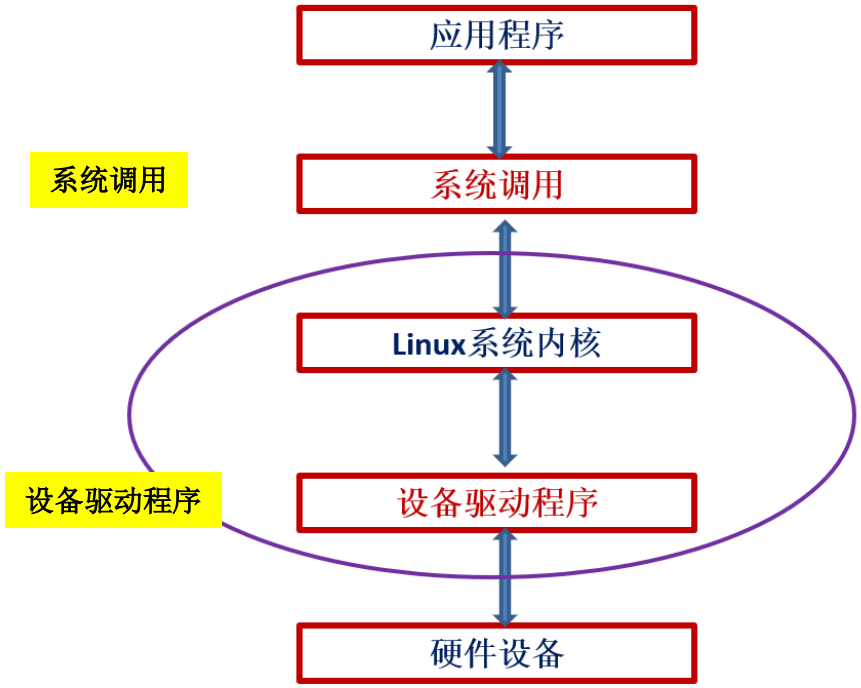
2023年11月7日

第9章　设备驱动程序设计基础

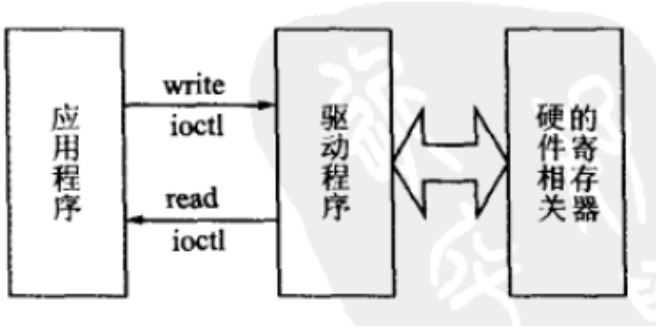
- 9.1　Linux设备驱动程序简介
- 9.2　设备驱动程序结构
- 9.3　Linux内核设备模型
- 9.4　内存映射和管理

9.1 Linux设备驱动程序简介

- 设备驱动程序与系统调用：



设备驱动程序与系统调用



设备驱动程序

- 系统调用：是操作系统内核（Linux系统内核）和应用程序之间的接口。

- 设备驱动程序：是操作系统内核（ Linux系统内核）和机器硬件之间的接口，设备驱动程序为应用程序屏蔽了硬件的细节，在应用程序看来，硬件设备只是一个设备文件，应用程序可以向操作普通文件一样对硬件设备进行操作。

- 设备驱动程序是内核的一部分，完成以下功能：
  ① 对设备的初始化和释放；
  ② 把数据从内核传送到硬件，和从硬件读取数据到内核；
  ③ 读取应用程序传送给设备文件的数据，和回送应用程序请求的数据；这需要在用户空间、内核空间、总线以及外设之间传输数据；
  ④ 检测和处理设备出现的错误。

- 设备驱动程序与应用程序的区别：

① 应用程序一般有一个main 函数，从头到尾执行一个任务。

② 设备驱动程序却不同，它没有main函数，通过使用宏module_init()，将初始化函数加入内核全局初始化函数列表中，在内核初始化时执行驱动的初始化函数，从而完成驱动的初始化和注册，之后驱动便停止等待被应用软件调用；驱动程序中有一个宏module_exit()注册退出处理函数，它在驱动退出时被调用。

```
module_init(gpio_uart485_init);
module_exit(gpio_uart485_init);
```

③ 应用程序可以和GLIBC 库连接，因此可以包含标准的头文件，比如<stdio.h>、<stdlib.h>。

④ 在设备驱动程序中是不能使用标准C 库的，因此不能调用所有的C 库函数，比如输出打印函数只能使用内核的printk 函数，包含的头文件只能是内核的头文件，比如<linux/module.h>。

```
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <pthread.h>
#include "tty.h"
#include "gprs.h"
```

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/device.h>
#include <asm/system.h>
#include <asm/uaccess.h>
#include <linux/platform_device.h>
#include <asm/irq.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_gpio.h>
```

- Linux 的设备驱动程序开发调试有两种方法：

  ① 一种是直接编译到内核，再运行新的内核来测试；

  ② 二是编译为模块的形式，单独加载运行调试。

- 第一种方法（直接编译到内核）效率较低，但在某些场合是唯一的方法。

- 第二种方式（编译为模块——模块方式）调试效率很高，它使用insmod 命令将编译的模块直接插入内核；如果出现故障，可以使用rmmod命令 从内核中卸载模块；不需要重新启动内核，这使驱动调试效率大大提高。lsmod命令为查看模块。

- **9.1.1 设备的分类**

  - 字符设备：无须缓冲直接读写的设备，如串口等设备。

  - 块设备：通过缓冲区进行（缓冲区通常为系统内存），只能以块为单位进行读写，块大小可以是512B或1024B，如硬盘等设备。

  - 网络设备：可以通过BSD套接口访问。

    - BSD（Berkeley Software Distribution，伯克利软件套件）是Unix的衍生系统，在1977至1995年间由加州大学伯克利分校开发和发布的。

• 9.1.2 设备文件

– Linux抽象了对硬件的处理，所有的硬件设备都可以作为普通文件一样对待，可以使用标准的系统调用接口来完成对设备的打开（open）、关闭（close）、读写（read、write）和I/O控制操作（ioctl），驱动程序的主要任务是实现这些系统调用函数。

```
com485 = open("/dev/UART485", O_RDWR);          //打开RS-485
close(com485);                                  //关闭RS-485
re = write(COMDevice,buf,strlen(buf));          //向RS-485写（发送）数据
re = read(COMDevice,buf,sizeof(buf));           //从RS-485读（接收）数据
ioctl(com485,UART485_TX);                       //设置RS-485为发送方式
ioctl(com485,UART485_RX);                       //设置RS-485为接收方式
```

– Linux系统中所有的硬件设备都使用一个特殊的设备文件来表示，如：
  - 系统中的第一个硬盘：/dev/had
  - 串口0：/dev/ttyS0
  - RS-485设备：/dev/485/0raw
  - CAN总线：/dev/can/0
  - A/D转换器：/dev/adc/0raw
  - 直流电机： /dev/dcm/0raw

– 对用户来说，设备文件和普通文件并无区别

– 查看设备文件命令：ls -l /dev

在Ubuntu的“终端”上执行查看设备文件命令：ls -l /dev

```
root@uptech-virtual-machine:/imx6/whzeng/nfc-3# ls -l /dev
总用量 0
crw-rw----  1 root video    10, 175 11月 20 07:53 agpgart
crw-------  1 root root     10, 235 11月 20 07:53 autofs
drwxr-xr-x  2 root root         640 11月 20 07:53 block
drwxr-xr-x  2 root root          80 11月 20 07:53 bsg
crw-------  1 root root     10, 234 11月 20 07:53 btrfs-control
drwxr-xr-x  3 root root          60 11月 20 07:53 bus
lrwxrwxrwx  1 root root           3 11月 20 07:53 cdrom -> sr0
drwxr-xr-x  2 root root        3580 11月 20 07:53 char
crw-------  1 root root      5,   1 11月 20 07:54 console
lrwxrwxrwx  1 root root          11 11月 20 07:53 core -> /proc/kcore
drwxr-xr-x  2 root root          60 11月 20 07:53 cpu
crw-------  1 root root     10,  60 11月 20 07:53 cpu_dma_latency
crw-------  1 root root     10, 203 11月 20 07:53 cuse
drwxr-xr-x  5 root root         100 11月 20 07:53 disk
crw-rw----+ 1 root audio    14,   9 11月 20 07:53 dmmidi
drwxr-xr-x  2 root root         100 11月 20 07:53 dri
crw-------  1 root root     10,  61 11月 20 07:53 ecryptfs
crw-rw----  1 root video    29,   0 11月 20 07:53 fb0
lrwxrwxrwx  1 root root          13 11月 20 07:53 fd -> /proc/self/fd
brw-rw----  1 root floppy    2,   0 11月 20 07:53 fd0
crw-rw-rw-  1 root root      1,   7 11月 20 07:53 full
crw-rw-rw-  1 root root     10, 229 11月 20 07:53 fuse
crw-------  1 root root    251,   0 11月 20 07:53 hidraw0
crw-------  1 root root     10, 228 11月 20 07:53 hpet
drwxr-xr-x  4 root root         220 11月 20 07:53 input
crw-r--r--  1 root root      1,  11 11月 20 07:53 kmsg
srw-rw-rw-  1 root root           0 11月 20 07:53 log
brw-rw----  1 root disk      7,   0 11月 20 07:53 loop0
brw-rw----  1 root disk      7,   1 11月 20 07:53 loop1
brw-rw----  1 root disk      7,   2 11月 20 07:53 loop2
brw-rw----  1 root disk      7,   3 11月 20 07:53 loop3
brw-rw----  1 root disk      7,   4 11月 20 07:53 loop4
brw-rw----  1 root disk      7,   5 11月 20 07:53 loop5
brw-rw----  1 root disk      7,   6 11月 20 07:53 loop6
brw-rw----  1 root disk      7,   7 11月 20 07:53 loop7
crw-------  1 root root     10, 237 11月 20 07:53 loop-control
crw-rw----  1 root lp        6,   0 11月 20 07:53 lp0
drwxr-xr-x  2 root root          60 11月 20 07:53 mapper
crw-------  1 root root     10, 227 11月 20 07:53 mcelog
crw-r-----  1 root kmem      1,   1 11月 20 07:53 mem
crw-------  1 root root     10,  57 11月 20 07:53 memory_bandwidth
crw-rw----+ 1 root audio    14,   2 11月 20 07:53 midi
drwxr-xr-x  2 root root          60 11月 20 07:53 net
```

在“超级终端Xshell”上执行查看设备文件命令：ls -l /dev

```
root@imx6dlsabresd:~# ls -l /dev
total 0
crw------- 1 root root     30,   0 Jan  1  1970 UART485
crw------- 1 root root     10, 235 Jan  1  1970 autofs
drwxr-xr-x 2 root root          640 Jan  1  1970 block
drwxr-xr-x 3 root root           60 Jan  1  1970 bus
drwxr-xr-x 2 root root         2700 Sep 29 08:03 char
crw------- 1 root root      5,   1 Sep 29 08:03 console
crw------- 1 root root     10,  61 Jan  1  1970 cpu_dma_latency
drwxr-xr-x 5 root root          100 Jan  1  1970 disk
drwxr-xr-x 2 root root           60 Jan  1  1970 dri
crw-rw---- 1 root video    29,   0 Jan  1  1970 fb0
crw-rw---- 1 root video    29,   1 Jan  1  1970 fb1
lrwxrwxrwx 1 root root           13 Jan  1  1970 fd -> /proc/self/fd
crw-rw-rw- 1 root root      1,   7 Jan  1  1970 full
crw-rw-rw- 1 root root     10, 229 Jan  1  1970 fuse
crw-rw---- 1 root video   199,   0 Jan  1  1970 galcore
crw------- 1 root root     10, 183 Jan  1  1970 hwrng
crw------- 1 root root     89,   0 Jan  1  1970 i2c-0
crw------- 1 root root     89,   1 Jan  1  1970 i2c-1
crw------- 1 root root     89,   2 Jan  1  1970 i2c-2
prw------- 1 root root           0 Jan  1  1970 initctl
drwxr-xr-x 3 root root         180 Jan  1  1970 input
```

- **9.1.3 主设备号和次设备号**

  - 主设备号：标识该设备的种类，也标识了该设备所使用的驱动程序，主设备号在/proc/devices文件中查看

    - 查看主设备号命令：**cat /proc/devices**

  - 次设备号：标识使用同一设备驱动程序的不同硬件设备

  - 创建设备文件的命令：**mknod /dev/lp0 c 6 0**
    - **/dev/lp0**：设备名
    - **c**：表示字符设备（**b**：表示块设备）
    - **6**：主设备号
    - **0**：次设备号

查看Ubuntu的主设备号
cat /proc/devices

Character devices: 字符设备

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/midi
14 sound/dmmidi
21 sg
29 fb
89 i2c
99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
216 rfcomm
226 drm
251 hidraw
252 bsg
253 watchdog
254 rtc

Block devices: 块设备

1 ramdisk
2 fd
259 blkext
7 loop
8 sd
9 md
11 sr
65 sd
66 sd
67 sd
68 sd
69 sd
70 sd
71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
252 device-mapper
253 virtblk
254 mdp

查看实验箱的主设备号

cat /proc/devices

Character devices: 字符设备

```
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 30 UART485
 81 video4linux
 89 i2c
 90 mtd
116 alsa
128 ptm
136 pts
180 usb
188 ttyUSB
189 usb_device
199 galcore
207 ttymxc
216 rfcomm
226 drm
247 mxc_vpu
248 ttyLP
249 mxc_hdmi
250 iio
251 mxc_ipu
252 ptp
253 pps
254 rtc
```

Block devices: 块设备

```
  1 ramdisk
259 blkext
  7 loop
  8 sd
 31 mtdblock
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
```

- **9.1.4 Linux设备驱动代码的分布**

  - 实验箱的所有设备驱动位于Ubuntu的：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/目录下

    - char：字符设备驱动
    - block：块设备驱动
    - pci：PCI驱动
    - scsi：SCSI驱动
    - net：网络驱动

/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/目录

- **9.1.5 Linux设备驱动程序的特点**

① 内核代码：设备驱动是内核代码的一部分。

② 内核接口：设备驱动必须为Linux内核提供一个标准接口。

③ 内核机制与服务：设备驱动可以使用标准的内核服务，如内存分配、中断和等待队列等。

④ 可加载：可以在需要的时候加载到内核（insmod），在不需要的时候从内核中卸载（rmmod）。

⑤ 可配置：设备驱动程序可以集成为内核的一部分，在编译内核时，可以选择把哪些驱动程序直接集成到内核里。

⑥ 动态性：系统启动或设备驱动初始化后，驱动程序将维护其控制的设备，即使该设备不存在，也不会影响整个系统的运行。

9.2 设备驱动程序结构

- Linux设备驱动程序与外界的接口分为以下三部分：
  ① 驱动程序与Linux操作系统内核的接口
  ② 驱动程序与系统引导的接口
  ③ 驱动程序与设备的接口

- Linux设备驱动程序的代码结构包括：
  ① 驱动程序的注册与注销
  ② 设备的打开与释放
  ③ 设备的读写操作
  ④ 设备的控制操作
  ⑤ 设备的中断和轮询处理

- **9.2.1 驱动程序的注册与注销**

  - 注册：赋予设备一个主设备号
    - 字符设备（chr）：register_chrdev_region()函数
      - 例如：retval = register_chrdev_region(devt,1,DRVNAME);
    - 块设备（blk）：register_blkdev_region()函数
  - 注销：释放占用的主设备号
    - 字符设备：unregister_chrdev_region()函数
      - 例如： unregister_chrdev_region(MKDEV(UART485_MAJOR,UART485_MINOR), 1);
    - 块设备：unregister_blkdev_region()函数

```
static int Uart485Init(void)
{
        dev_t devt;
        int retval;

        devt = MKDEV(UART485_MAJOR,UART485_MINOR);
        retval = register_chrdev_region(devt,1,DRVNAME);

        if(retval>0)
                return retval;

        cdev_init(&uart485cdev,&uart485_fops);
        retval = cdev_add(&uart485cdev,devt,1);
```

RS-485设备的注册

```
static void Uart485Exit(void)
{
        device_destroy(uart485_class, MKDEV(UART485_MAJOR,UART485_MINOR));
        class_destroy(uart485_class);
        cdev_del(&uart485cdev);
        unregister_chrdev_region(MKDEV(UART485_MAJOR,UART485_MINOR), 1);
}
```

RS-485设备的注销

• 9.2.2 设备的打开与释放

– file_operations结构体：设备文件操作结构体

```
static const struct file_operations uart485_fops = {
        .owner = THIS_MODULE,
        .write = Uart485PowerWrite,
        .read = Uart485PowerRead,
        .open = Uart485PowerOpen,
        .unlocked_ioctl = Uart485PowerIoctl,
};
```

RS-485设备的file_operations结构体

– 设备的打开：通过调用file_operations结构体中的open()函数完成
  - 例如： com485 = open("/dev/UART485", O_RDWR);　　//打开RS-485

– 设备的释放（关闭）：通过调用file_operations结构体中的release()函数完成（有时也称为close()函数）
  - 例如： close(com485);　　//关闭RS-485

• 9.2.3 设备的读写操作

– 设备的读操作：通过调用file_operations结构体中的read()函数完成

- 字符设备：read()函数
  - 例如：re = read(COMDevice,buf,sizeof(buf)); //从RS-485读（接收）数据
- 块设备：block_read()

– 设备的写操作：通过调用file_operations结构体中的write()函数完成

- 字符设备：write()函数
  - 例如：re = write(COMDevice,buf,strlen(buf)); //向RS-485写（发送）数据
- 块设备：block_write()函数

• 9.2.4 设备的控制操作

- 设备的控制操作：通过调用file_operations结构体中的ioctl()函数完成

- 例如，使RS-485处于发送模式或接收模式：
  - ioctl(com485,UART485_TX);　　//设置RS-485为发送方式（TX）
  - ioctl(com485,UART485_RX);　　//设置RS-485为接收方式（RX）

• 9.2.5 设备的轮询和中断处理

– 轮询方式（查询方式）：对于不支持中断的硬件设备，读写时需要轮流查询设备状态，以便决定是否继续进行数据传输
  - 轮询设备驱动可以通过使用系统定时器，使内核周期性的调用设备驱动中的某个例程来检查设备状态

– 中断方式：内核负责把硬件产生的中断传递给相应的设备驱动
  - 在/proc/interrupts文件中可以看到设备驱动所对应的中断号及类型
  - 查询中断号的命令：cat /proc/interrupts

Ubuntu的中断号

```
root@uptech-virtual-machine:/imx6/whzeng/nfc-3# cat   /proc/interrupts
           CPU0
  0:         50   IO-APIC-edge      timer
  1:       2272   IO-APIC-edge      i8042
  6:          2   IO-APIC-edge      floppy
  7:          0   IO-APIC-edge      parport0
  8:          1   IO-APIC-edge      rtc0
  9:          0   IO-APIC-fasteoi   acpi
 12:      96144   IO-APIC-edge      i8042
 14:          0   IO-APIC-edge      ata_piix
 15:          0   IO-APIC-edge      ata_piix
 16:       5445   IO-APIC  16-fasteoi   vmwgfx, snd_ens1371, eth0
 17:      74235   IO-APIC  17-fasteoi   ehci_hcd:usb1, ioc0
 18:        165   IO-APIC  18-fasteoi   uhci_hcd:usb2
 24:          0   PCI-MSI-edge      PCIe PME, pciehp
 25:          0   PCI-MSI-edge      PCIe PME, pciehp
 26:          0   PCI-MSI-edge      PCIe PME, pciehp
 27:          0   PCI-MSI-edge      PCIe PME, pciehp
 28:          0   PCI-MSI-edge      PCIe PME, pciehp
 29:          0   PCI-MSI-edge      PCIe PME, pciehp
 30:          0   PCI-MSI-edge      PCIe PME, pciehp
 31:          0   PCI-MSI-edge      PCIe PME, pciehp
 32:          0   PCI-MSI-edge      PCIe PME, pciehp
 33:          0   PCI-MSI-edge      PCIe PME, pciehp
 34:          0   PCI-MSI-edge      PCIe PME, pciehp
 35:          0   PCI-MSI-edge      PCIe PME, pciehp
 36:          0   PCI-MSI-edge      PCIe PME, pciehp
 37:          0   PCI-MSI-edge      PCIe PME, pciehp
 38:          0   PCI-MSI-edge      PCIe PME, pciehp
 39:          0   PCI-MSI-edge      PCIe PME, pciehp
 40:          0   PCI-MSI-edge      PCIe PME, pciehp
 41:          0   PCI-MSI-edge      PCIe PME, pciehp
 42:          0   PCI-MSI-edge      PCIe PME, pciehp
 43:          0   PCI-MSI-edge      PCIe PME, pciehp
 44:          0   PCI-MSI-edge      PCIe PME, pciehp
 45:          0   PCI-MSI-edge      PCIe PME, pciehp
 46:          0   PCI-MSI-edge      PCIe PME, pciehp
 47:          0   PCI-MSI-edge      PCIe PME, pciehp
 48:          0   PCI-MSI-edge      PCIe PME, pciehp
 49:          0   PCI-MSI-edge      PCIe PME, pciehp
 50:          0   PCI-MSI-edge      PCIe PME, pciehp
 51:          0   PCI-MSI-edge      PCIe PME, pciehp
 52:          0   PCI-MSI-edge      PCIe PME, pciehp
 53:          0   PCI-MSI-edge      PCIe PME, pciehp
```

实验箱的中断号

```
root@imx6dlsabresd:~# cat   /proc/interrupts
           CPU0       CPU1
 29:     166850      14973         GIC  29  twd
 34:          0          0         GIC  34  sdma
 35:          0          0         GIC  35  VPU_JPG_IRQ
 37:          0          0         GIC  37  2400000.ipu
 38:          7          0         GIC  38  2400000.ipu
 41:        707          0         GIC  41
 44:          0          0         GIC  44  VPU_CODEC_IRQ
 50:          0          0         GIC  50  vdoa
 51:          0          0         GIC  51  rtc alarm
 52:          0          0         GIC  52  snvs-secvio
 56:          0          0         GIC  56  mmc2
 57:       2468          0         GIC  57  mmc3
 58:        521          0         GIC  58  2020000.serial
 68:        132          0         GIC  68  21a0000.i2c
 69:       1491          0         GIC  69  21a4000.i2c
 70:      76809          0         GIC  70  21a8000.i2c
 72:       1647          0         GIC  72  2184200.usb
 75:          0          0         GIC  75  2184000.usb
```

9.3 Linux内核设备模型

- **9.3.1 设备模型建立的目的**

  - 内核设备模型是为了适应系统拓扑结构越来越复杂，对电源管理、热插拔支持要求越来越高等形势下开发的全新的设备模型，它采用sysfs文件系统，其作用是将系统中的设备组织成层次结构，然后向用户程序提供内核数据结构信息。

  - 设备模型提供独立的机制表示设备，并表示其在系统中的拓扑结构。

- **9.3.2 sysfs——设备拓扑结构的文件系统表现**

  - 将设备结构树导出为一个文件系统，即sysfs文件系统，sysfs文件系统挂载在“/sys”目录下

  - “/sys/devices”目录将设备模型导出到用户空间，其目录结构就是系统中实际的设备拓扑结构

```
root@imx6dlsabresd:/sys# ls
block  bus  class  dev  devices
firmware  fs  fsl_otp  kernel  module
power
root@imx6dlsabresd:/sys#
```

实验箱的“/sys”目录

```
root@imx6dlsabresd:/sys/devices# ls
ARMv7 Cortex-A9  breakpoint
platform  soc0  software  system
virtual
root@imx6dlsabresd:/sys/devices#
```

实验箱的“/sys/devices”目录

• 9.3.3 驱动模型和sysfs

– Linux 设备驱动模型的基本元素是：

① 总线类型（总线结构）：bus，位于/sys/bus

② 设备（设备结构）：devices，位于/sys/devices

```
root@imx6dlsabresd:/sys/bus# ls
clockevents  cpu        i2c     mmc     sdio  spi
workqueue
clocksource  event_source  iio     platform  serio  usb
container    hid        mdio_bus  scsi    soc   usb-serial
root@imx6dlsabresd:/sys/bus#
```

```
root@imx6dlsabresd:/sys/devices# ls
ARMv7 Cortex-A9  breakpoint  platform  soc0
software  system  virtual
root@imx6dlsabresd:/sys/devices#
```

③ 设备类别（设备类结构）：class，位于/sys/class

④ 设备驱动（驱动结构）：drivers，例如，USB设备的驱动位于/sys/bus/usb/drivers

```
root@imx6dlsabresd:/sys/class# ls
UART485   drm          lcd      mxc_ipu      rtc        udc
ata_device  firmware     leds     mxc_vpu      scsi_device  vc
ata_link  gpio         ledtest  net          scsi_disk  video4linux
ata_port  graphics     mdio_bus  power_supply  scsi_host  vtconsole
backlight  graphics_class  mem      pps          sound
bdi       hwmon        misc     ptp          spi_master
block     i2c-dev      mmc_host  pwm          thermal
bluetooth  ieee80211     mtd      rc           tty
dma       input        mxc_hdmi  regulator    ubi
root@imx6dlsabresd:/sys/class#
```

```
root@imx6dlsabresd:/sys/bus/usb/drivers# ls
ath3k   btusb  option   usb       usbfs
usbserial
bcm203x  hub   rtl8188fu  usb-storage  usbhid
usbserial_generic
root@imx6dlsabresd:/sys/bus/usb/drivers#
```

• 9.3.4 platform总线

– platform总线（平台总线）是Linux内核中的一个虚拟总线，使设备的管理更加简单化，目前大部分的驱动都是用platform总线来写的。

– platform总线分为以下几个部分：

① platform_bus

② platform_device

③ platform_driver

```
root@imx6dlsabresd:/sys/bus# ls
clockevents  cpu          i2c      mmc      sdio   spi
workqueue
clocksource  event_source  iio      platform  serio  usb
container    hid          mdio_bus  scsi     soc    usb-serial
root@imx6dlsabresd:/sys/bus#
```

```
root@imx6dlsabresd:/sys/bus/platform# ls
devices  drivers  drivers_autoprobe
drivers_probe  uevent
root@imx6dlsabresd:/sys/bus/platform#
```

9.4 内存映射和管理

• 9.4.1 物理地址映射到虚拟地址

– 在内核中访问I/O内存（I/O与内存统一编制，访问I/O就像访问内存一样）之前，我们只有I/O内存的物理地址，这样是无法通过软件直接访问的，需要首先用ioremap()函数将设备所处的物理地址映射到内核虚拟地址空间（3GB~4GB），然后，才能根据映射所得到的内核虚拟地址范围，通过访问指令访问这些I/O内存资源。

- void * ioremap(unsigned long phys_addr, unsigned long size, unsigned long flags)
  - phys_addr：要映射的起始的I/O地址
  - size：要映射的空间的大小
  - flags：要映射的I/O空间的和权限有关的标志

• 9.4.2 内核空间映射到用户空间

– 使用mmap系统调用，可以将内核空间的地址映射到用户空间

- void* mmap(void* start, size_t length, int prot, int flags, int fd, off_t offset);
- int (* mmap)(struct file*filp, struct vm_area_struct *vma);

– 查看设备内存是如何映射的： cat /proc/iomem

```
root@imx6dlsabresd:/sys/bus/platform# cat /proc/iomem
00110000-00111fff : /soc/dma-apbh@00110000
00120000-00128fff : 120000.hdmi_core
00130000-00133fff : galcore register region
00134000-00137fff : galcore register region
00905000-0091ffff : /soc/sram@00905000
02020000-02023fff : /soc/aips-bus@02000000/spba-bus@02000000/serial@02020000
02034000-02037fff : /soc/aips-bus@02000000/spba-bus@02000000/asrc@02034000
02080000-02083fff : /soc/aips-bus@02000000/pwm@02080000
02084000-02087fff : /soc/aips-bus@02000000/pwm@02084000
02088000-0208bfff : /soc/aips-bus@02000000/pwm@02088000
0208c000-0208ffff : /soc/aips-bus@02000000/pwm@0208c000
02090000-02093fff : /soc/aips-bus@02000000/can@02090000
0209c000-0209ffff : /soc/aips-bus@02000000/gpio@0209c000
020a0000-020a3fff : /soc/aips-bus@02000000/gpio@020a0000
020a4000-020a7fff : /soc/aips-bus@02000000/gpio@020a4000
020a8000-020abfff : /soc/aips-bus@02000000/gpio@020a8000
020ac000-020affff : /soc/aips-bus@02000000/gpio@020ac000
```

- **LED点阵：**

```
cpld = (unsigned short*)mmap(NULL,(size_t)0x20,PROT_READ | PROT_WRITE |
PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
```

- **步进电机：**

```
cpld = (unsigned char*)mmap(NULL,(size_t)0x04,PROT_READ | PROT_WRITE |
PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
```

- **八段数码管：**

```
cpld = (unsigned char*)mmap(NULL,(size_t)0x10,PROT_READ | PROT_WRITE |
PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
```

- **交通灯：**

```
cpld = (unsigned char*)mmap(NULL,(size_t)0x4,PROT_READ | PROT_WRITE |
PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
```

- **LCD液晶显示器：**

```
frame_base = mmap(NULL, BUFFER_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fb_con, 0);
```

小结

- **Linux设备驱动程序**
  - 设备驱动程序与系统调用，设备驱动程序与应用程序的区别，Linux 的设备驱动程序开发调试有两种方法（直接编译到内核，编译为模块），设备的分类（字符设备、块设备、网络设备），设备文件，主设备号和次设备号
- **Linux设备驱动程序结构**
  - 驱动程序的注册与注销，设备的打开与释放，设备的读写操作，设备的控制操作，设备的中断和轮询处理
- **Linux内核设备模型**
  - **sysfs文件系统，platform总线（平台总线）**
- 内存映射和管理
  - 物理地址映射到虚拟地址，内核空间映射到用户空间

进一步探索

① RS-232驱动程序：在Linux内核中

② 八段数码管驱动程序：内存映射（mmap）

③ LED点阵驱动程序：内存映射（mmap）

④ 步进电机驱动程序：内存映射（mmap）

⑤ 红外对射传感器驱动程序：采用RS-232方式，在Linux内核中

⑥ NFC模块驱动程序：采用RS-232方式，在Linux内核中

- 阅读设备驱动程序的源代码

① RS-485驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/char/uptech485.c

② CAN总线驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/net/can/flexcan.c

③ 小键盘驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/keyboard/gpio_keys.c

④ LED灯驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/char/imx6-leds.c

⑤ LCD液晶显示器驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/video/console/fbcon.c、/home/uptech/fsl-6dl-source/kernel-4.9.88/include/linux/fb.h、/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/video/fbdev/core/fbmem.c

⑥ 蜂鸣器驱动程序：/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/misc/gpio-beeper.c

⑦ 摄像头的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/media/platform/mxc/capture目录下ov5640*.c文件

⑧ 陀螺仪的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/misc/mpu6880/mpu6880.c

Thanks