《嵌入式系统》

（第九讲）

厦门大学信息学院软件工程系 曾文华

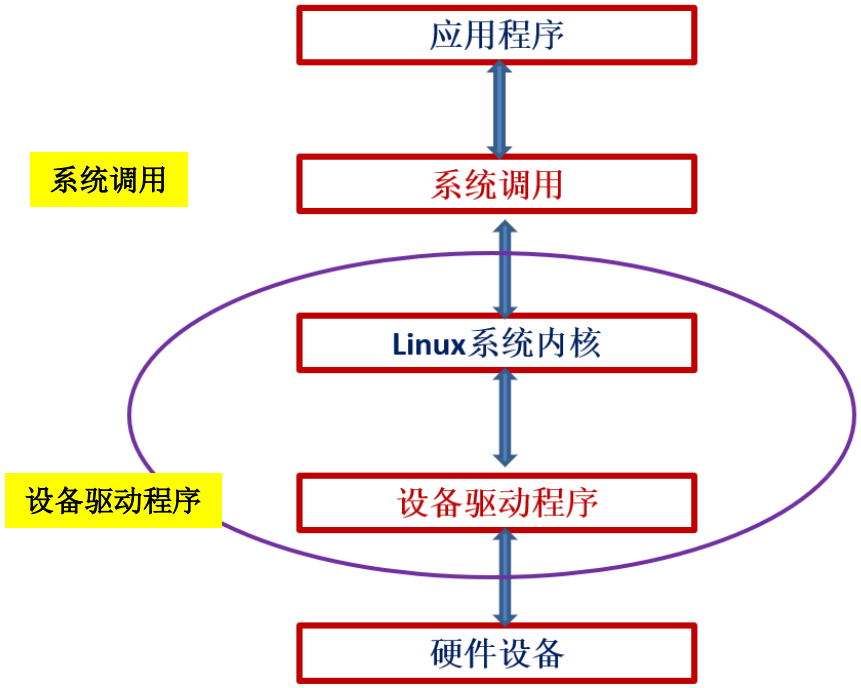
2024年11月5日

第9章　设备驱动程序设计基础

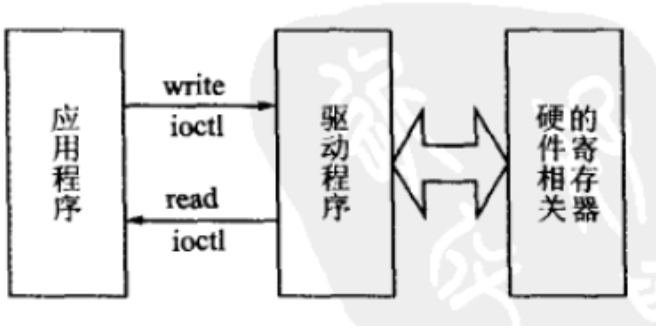
- 9.1　Linux设备驱动程序简介
- 9.2　设备驱动程序结构
- 9.3　Linux内核设备模型
- 9.4　内存映射和管理

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(farsight_dev_init);
module_exit(farsight_dev_exit);
```

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 <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.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。

```
insmod character.ko        rmmod character
```

- **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。

- fd = open(/dev/character_device, O_RDWR); //打开设备文件
- close(fd); //关闭设备文件
- ret = write(fd,"APP test",8); //写操作
- ret = read(fd, buf, 5); //读操作
- ret = ioctl(fd,'a',0); //ioctl 操作

– Linux系统中所有的硬件设备都使用一个特殊的设备文件（设备名）来表示，如：

- 基于GPIO的LED灯：/dev/leds_ctl
- 蜂鸣器：/dev/buzzer_ctl
- 按键：/dev/farsight_keys
- 直流电机：/dev/dc_motor

– 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

```
linux@linux-pc:/$ ls -l /dev
total 0
crw-------  1 root  root     10, 175 10月 28 15:11 agpgart
crw-r--r--  1 root  root     10, 235 10月 28 15:11 autofs
drwxr-xr-x  2 root  root          620 10月 28 15:12 block
drwxr-xr-x  2 root  root           80 10月 28 15:11 bsg
crw-------  1 root  root     10, 234 10月 28 15:11 btrfs-control
drwxr-xr-x  3 root  root           60 10月 28 15:11 bus
lrwxrwxrwx  1 root  root            3 10月 28 15:11 cdrom -> sr0
lrwxrwxrwx  1 root  root            3 10月 28 15:11 cdrw -> sr0
drwxr-xr-x  2 root  root         3740 10月 28 20:14 char
crw-------  1 root  root      5,    1 10月 28 15:12 console
lrwxrwxrwx  1 root  root           11 10月 28 15:11 core -> /proc/kcore
crw-------  1 root  root     10,   60 10月 28 15:11 cpu_dma_latency
crw-------  1 root  root     10, 203 10月 28 15:11 cuse
drwxr-xr-x  6 root  root          120 10月 28 15:11 disk
brw-rw----  1 root  disk    253,    0 10月 28 15:11 dm-0
brw-rw----  1 root  disk    253,    1 10月 28 15:11 dm-1
brw-rw----  1 root  disk    253,    2 10月 28 15:11 dm-2
brw-rw----  1 root  disk    253,    3 10月 28 15:11 dm-3
crw-rw----+ 1 root  audio    14,    9 10月 28 15:12 dmmidi
drwxr-xr-x  3 root  root          100 10月 28 15:11 dri
lrwxrwxrwx  1 root  root            3 10月 28 15:11 dvd -> sr0
crw-------  1 root  root     10,   62 10月 28 15:11 ecryptfs
crw-rw----  1 root  video    29,    0 10月 28 15:11 fb0
lrwxrwxrwx  1 root  root           13 10月 28 15:11 fd -> /proc/self/fd
crw-rw-rw-  1 root  root      1,    7 10月 28 15:11 full
crw-rw-rw-  1 root  root     10, 229 10月 28 15:11 fuse
crw-------  1 root  root    242,    0 10月 28 15:11 hidraw0
crw-------  1 root  root     10, 228 10月 28 15:11 hpet
drwxr-xr-x  2 root  root            0 10月 28 15:11 hugepages
```

在实验箱的“Xshell”上执行查看设备文件命令：ls -l /dev

```
linux@localhost:~$ ls -l /dev
total 4
crw-------  1 root  root     10,  55 Feb 11 16:28 binder
drwxr-xr-x  2 root  root          420 Feb 11 16:28 block
drwxr-xr-x  3 root  root           60 Jan  1  1970 bus
crwxrwxrwx  1 root  root     10,  60 Feb 11 16:28 buzzer
crwxrwxrwx  1 root  root     10,  58 Feb 11 16:28 buzzer_ctl
crw-------  1 root  root    250,   0 Feb 11 16:28 cec0
drwxr-xr-x  2 root  root         3640 Feb 11 16:28 char
crw-------  1 root  root      5,   1 Feb 11 16:28 console
crw-------  1 root  root     10,  51 Feb 11 16:28 cpu_dma_latency
crwxrwxrwx  1 root  root     10,  56 Feb 11 16:28 dc_motor
drwxr-xr-x  7 root  root          140 Feb 11 16:28 disk
drwxr-xr-x  2 root  root          100 Jan  1  1970 dri
crwxrwxrwx  1 root  root     10,  57 Feb 11 16:28 farsight_keys
crw-rw----  1 root  video    29,   0 Feb 11 16:28 fb0
lrwxrwxrwx  1 root  root           13 Feb 11 16:28 fd -> /proc/self/fd
crw-rw-rw-  1 root  root      1,   7 Feb 11 16:28 full
crw-rw-rw-  1 root  root     10, 229 Feb 11 16:28 fuse
crw-rw-rw-  1 root  root     10,  46 Feb 11 16:28 hdmi_hdcp1x
crw-------  1 root  root     10,  54 Feb 11 16:28 hwbinder
crw-------  1 root  root     10, 183 Feb 11 16:28 hwrng
crw-------  1 root  root     89,   0 Feb 11 16:28 i2c-0
crw-------  1 root  root     89,   1 Feb 11 16:28 i2c-1
crwxrwxrwx  1 root  root     89,   4 Feb 11 16:28 i2c-4
crw-------  1 root  root     89,   9 Feb 11 16:28 i2c-9
```

- **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

```
linux@linux-pc:/$ 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/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
204 ttyMAX
216 rfcomm
226 drm
241 aux
242 hidraw
243 vfio
244 bsg
245 watchdog
246 ptp
247 pps
248 cec
249 rtc
250 dax
251 dimmctl
252 ndctl
253 tpm
254 gpiochip
```

字符设备

```
Block devices:
  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
253 device-mapper
254 mdp
259 blkext
linux@linux-pc:/$
```

块设备

查看实验箱的主设备号

cat /proc/devices

```
linux@localhost:~$ cat   /proc/devices
Character devices:
  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
 81 video4linux
 89 i2c
 90 stepmotor
108 ppp
116 alsa
128 ptm
136 pts
401 led
153 spi
166 ttyACM
180 usb
188 ttyUSB
189 usb_device
226 drm
239 hidraw
240 ttyGS
241 usbmon
242 nvme
243 leds_ctl
244 rkvdec
245 vpu_service
500 rfid
500 rfid
500 rfid
500 rfid
500 rfid
246 bsg
247 iio
248 ptp
249 pps
250 cec
251 media
252 rtc
253 tpm
254 ttyFIQ
```

字符设备

```
Block devices:
  1 ramdisk
259 blkext
  7 loop
  8 sd
 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
179 mmc
253 nvme
254 zram
linux@localhost:~$
```

块设备

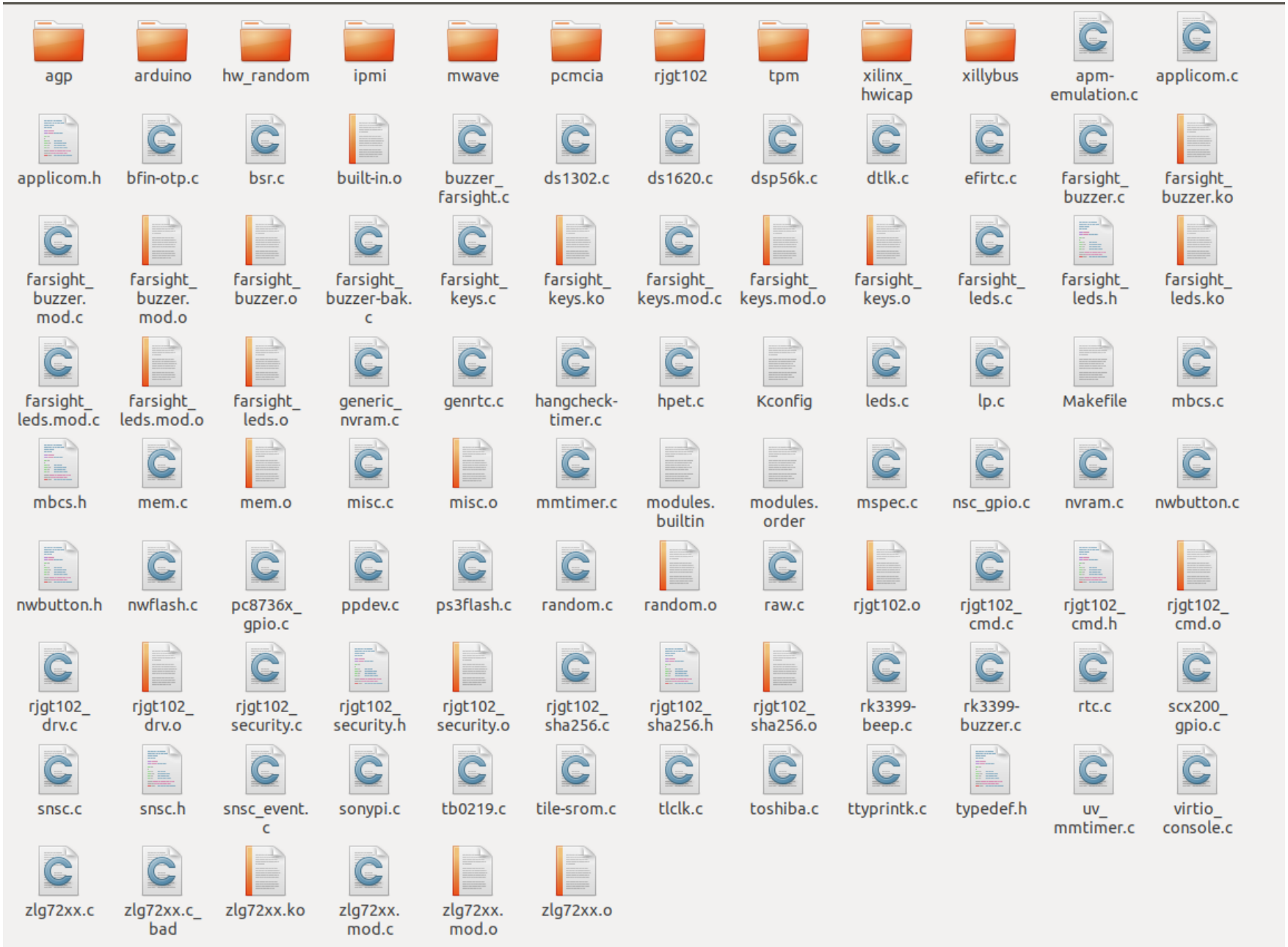
- **9.1.4 Linux设备驱动代码的分布**

  - 实验箱的所有设备驱动位于Ubuntu的：/home/linux/workdir/fs3399/system/kernel/drivers/目录下

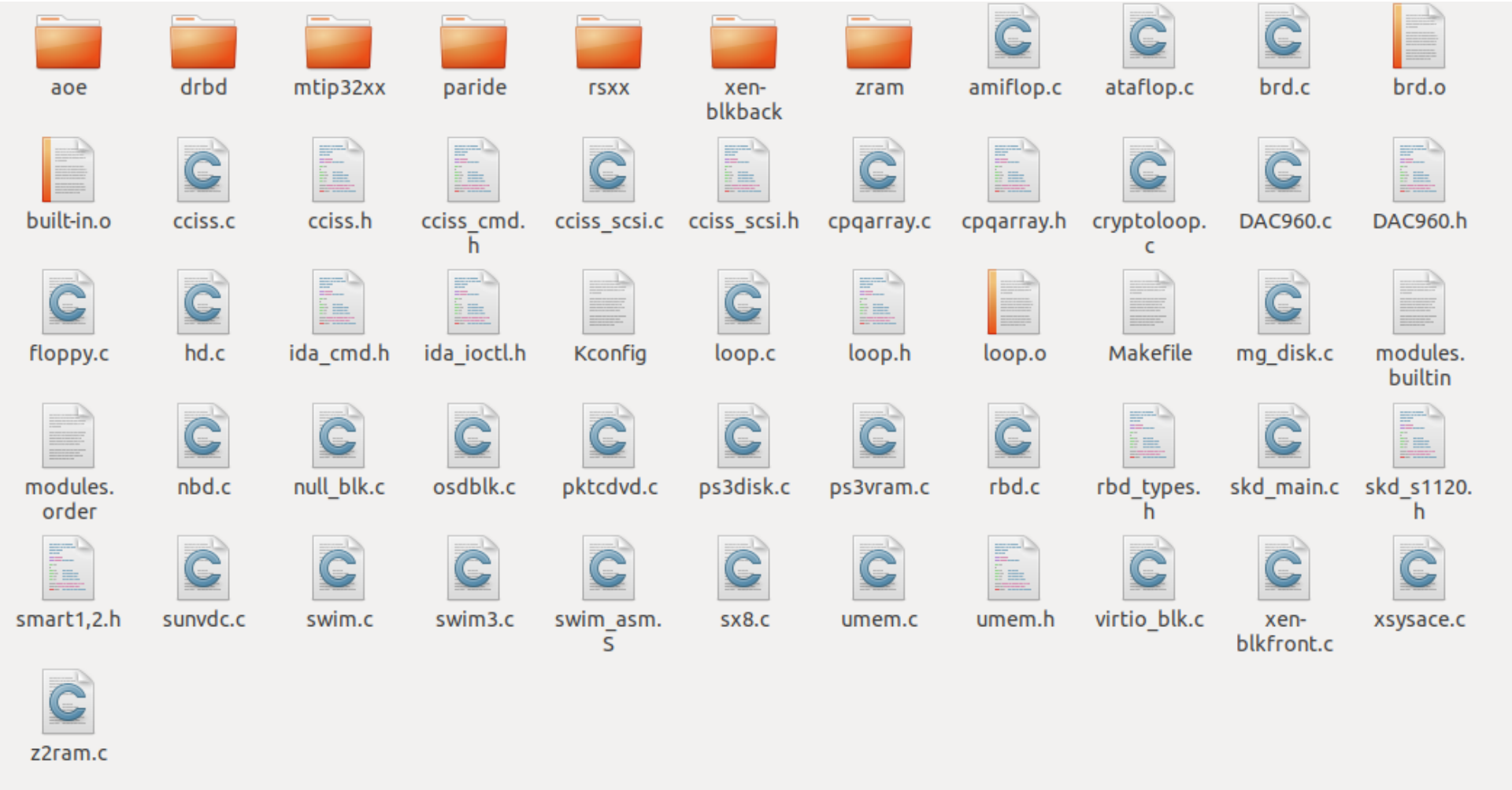
    - char：字符设备驱动
    - block：块设备驱动
    - pci：PCI驱动
    - scsi：SCSI驱动
    - net：网络驱动

/home/linux/workdir/fs3399/system/kernel/drivers/目录

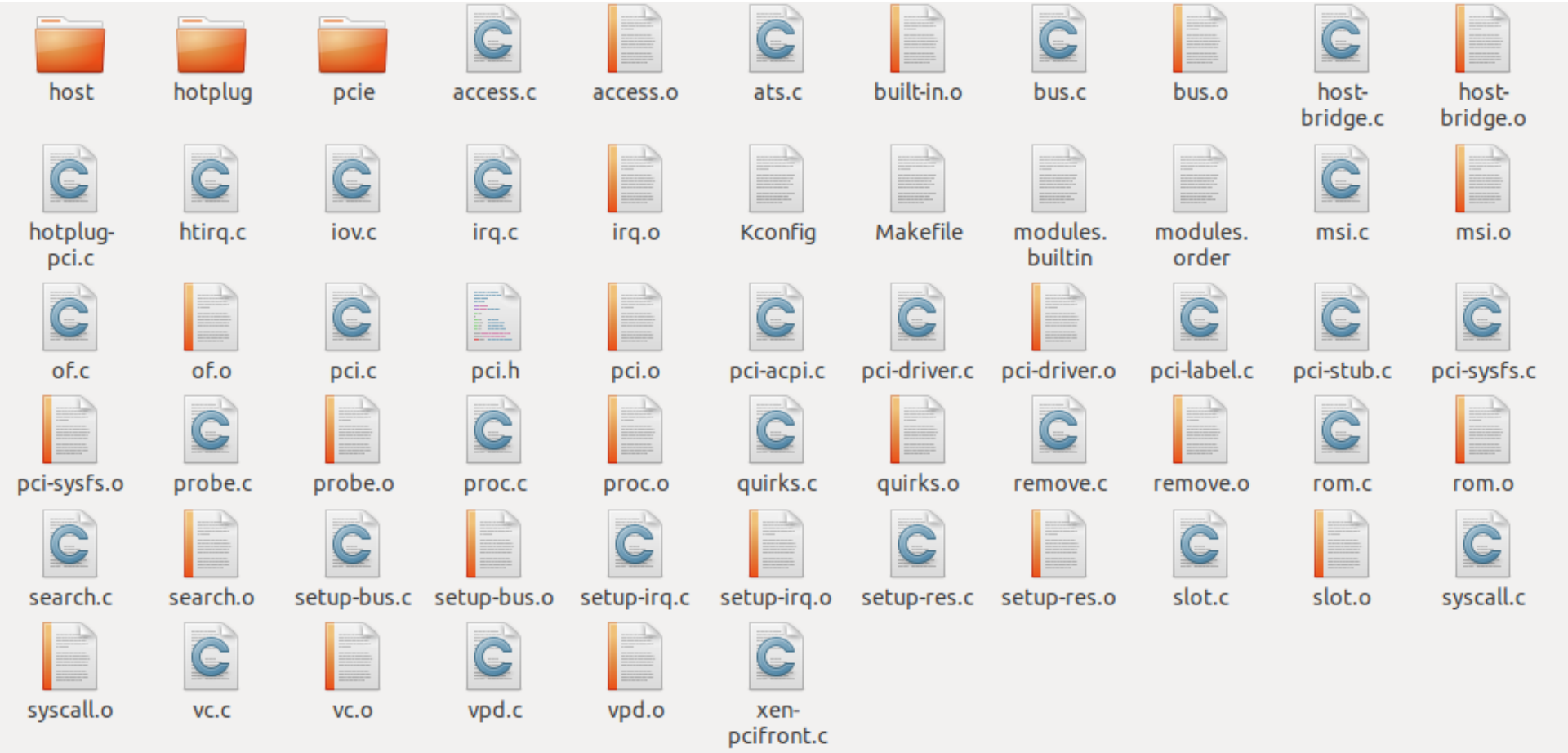
/home/linux/workdir/fs3399/system/kernel/drivers/char目录



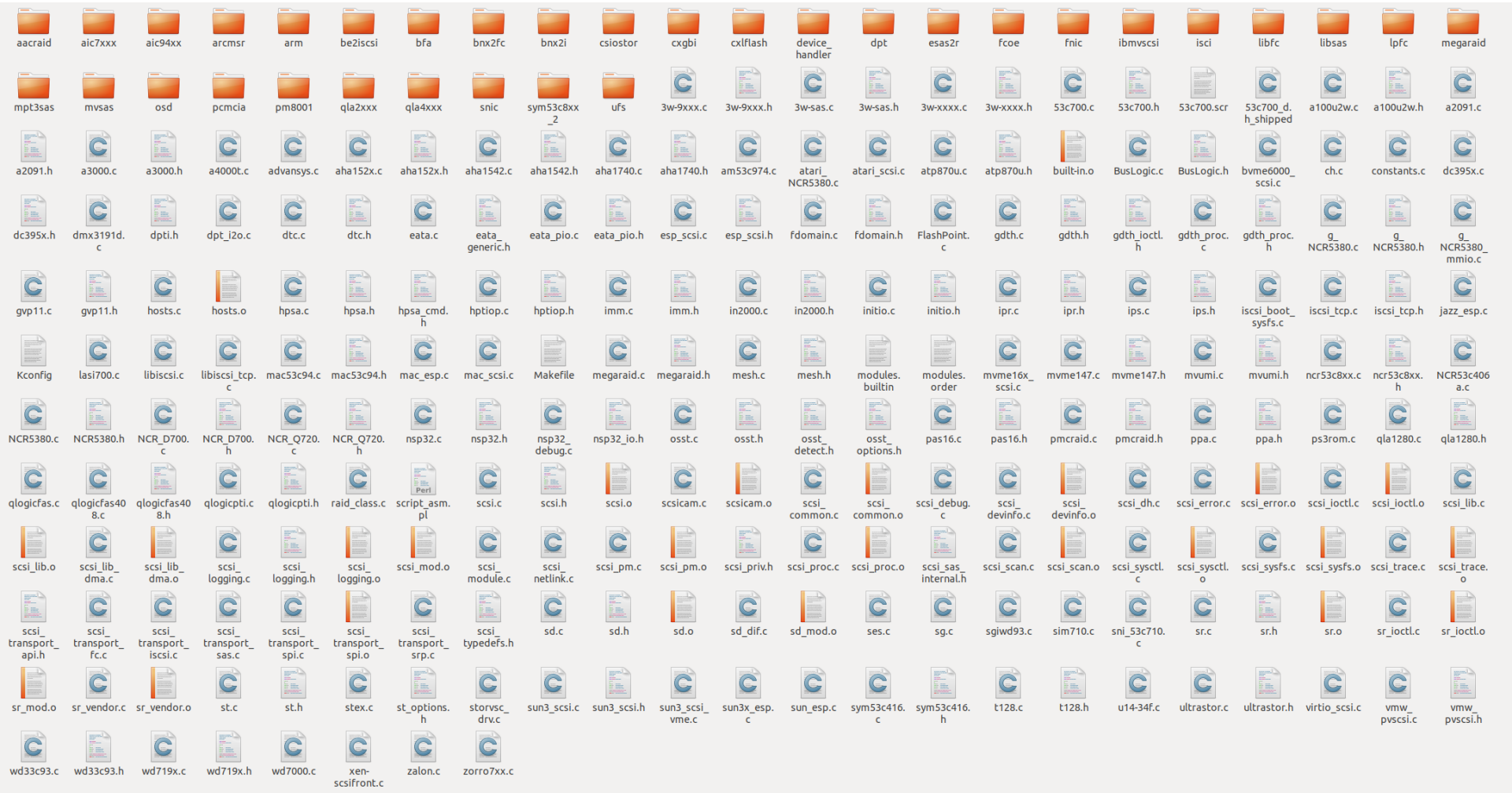
/home/linux/workdir/fs3399/system/kernel/drivers/block目录



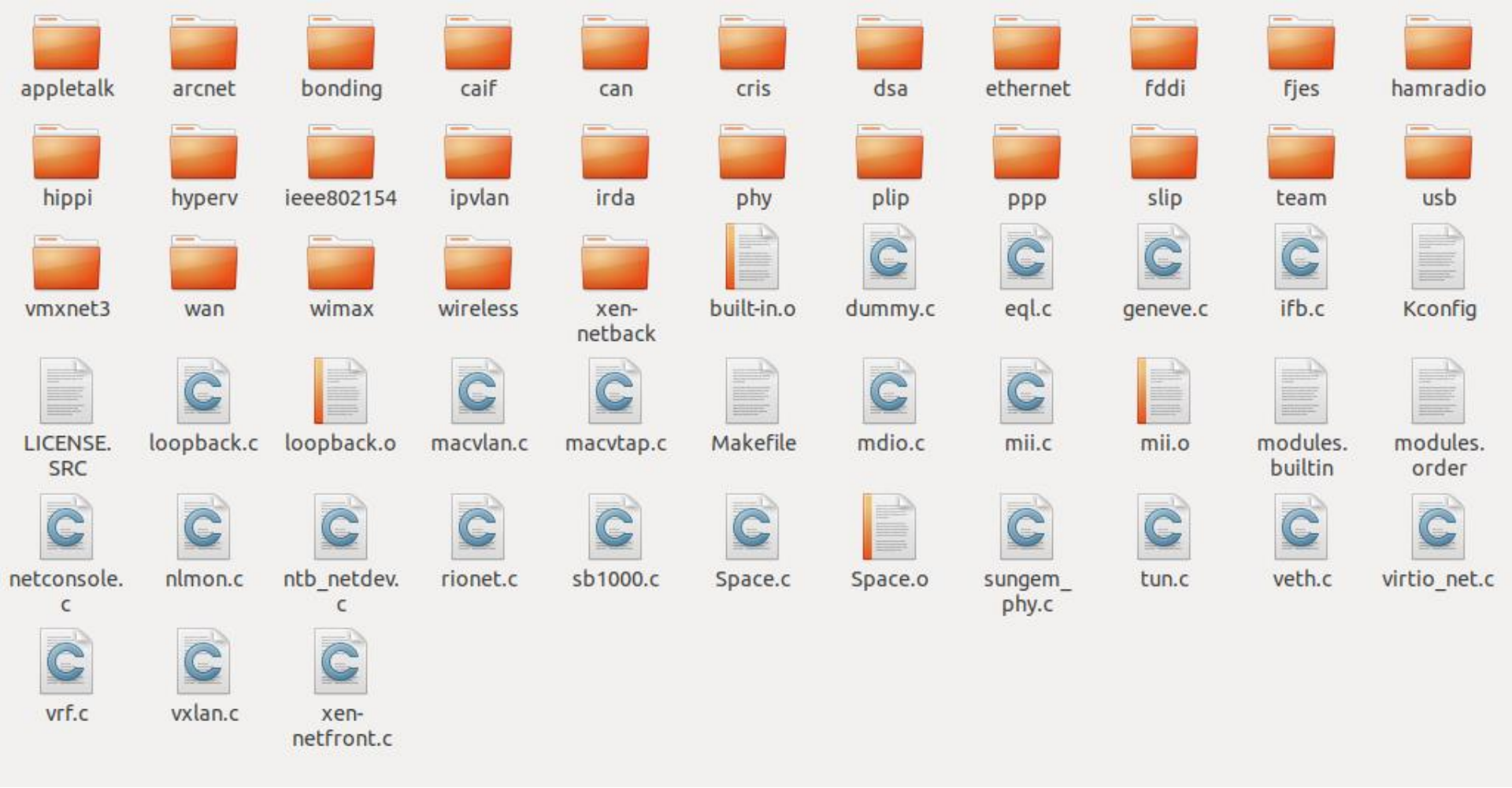
/home/linux/workdir/fs3399/system/kernel/drivers/pci目录



/home/linux/workdir/fs3399/system/kernel/drivers/scsi目录



/home/linux/workdir/fs3399/system/kernel/drivers/net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(devnum,1);
    - 块设备：unregister_blkdev_region()函数

• 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 struct file_operations farsight_ops =
{
        .owner          = THIS_MODULE,
        .open           = farsight_open,
        .read           = farsight_read,
        .write          = farsight_write,
        .release        = farsight_close,
        .unlocked_ioctl = farsight_ioctl,
};
```

字符设备character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： fd = open("/dev/dc_motor", O_RDWR); //打开直流电机

– 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(fd); //关闭直流电机

实验箱的几个设备驱动程序的file_operations结构体

```
static struct file_operations farsight_ops =                    //设备文件结构体
{
        .owner          = THIS_MODULE,
        .open           = farsight_open,
        .read           = farsight_read,
        .write          = farsight_write,
        .release        = farsight_close,
        .unlocked_ioctl = farsight_ioctl,
};
```

字符设备

```
static struct file_operations farsight_led_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

LED灯

```
static struct file_operations farsight_led_ops = {              //设备文件结构体
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

```
static struct file_operations farsight_pwm_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

呼吸灯

```
static struct file_operations farsight_buzzer_ops = {
    .owner          = THIS_MODULE,
    .open           = farsight_buzzer_open,
    .release        = farsight_buzzer_close,
    .unlocked_ioctl = farsight_buzzer_ioctl,
};
```

蜂鸣器

```
static struct file_operations farsight_keys_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_keys_open,
        .release        = farsight_keys_close,
        .read           = farsight_keys_read,
};
```

按键

```
static struct file_operations zlg72XX_fops = {
        .owner = THIS_MODULE,
        .open = zlg72XX_open,
        .release = zlg72XX_release,
        .unlocked_ioctl = zlg72XX_ioctl,
};
```

小键盘/数码管

• 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： ret = read(fd, buf, 5); //字符设备character的读操作
- 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： ret = write(fd,"APP test",8); //字符设备character的写操作
- 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：
    - ret = ioctl(fd,'a',0);　　//字符设备character的ioctl 操作
    - ret = ioctl(fd,'b',0);　　//字符设备character的ioctl 操作

• 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

```
linux@linux-pc:/$ cat   /proc/interrupts
           CPU0       CPU1
  0:          5          0   IO-APIC    2-edge      timer
  1:        928          0   IO-APIC    1-edge      i8042
  8:          0          1   IO-APIC    8-edge      rtc0
  9:          0          0   IO-APIC    9-fasteoi   acpi
 12:          0     130755   IO-APIC   12-edge      i8042
 14:          0          0   IO-APIC   14-edge      ata_piix
 15:          0          0   IO-APIC   15-edge      ata_piix
 16:      31077       1791   IO-APIC   16-fasteoi   vmwgfx, snd_ens1371
 17:      28809       3413   IO-APIC   17-fasteoi   ehci_hcd:usb1, ioc0
 18:          8        166   IO-APIC   18-fasteoi   uhci_hcd:usb2
 19:       5076       1131   IO-APIC   19-fasteoi   ens33
 24:          0          0   PCI-MSI 344064-edge      PCIe PME, pciehp
 25:          0          0   PCI-MSI 346112-edge      PCIe PME, pciehp
 26:          0          0   PCI-MSI 348160-edge      PCIe PME, pciehp
 27:          0          0   PCI-MSI 350208-edge      PCIe PME, pciehp
 28:          0          0   PCI-MSI 352256-edge      PCIe PME, pciehp
 29:          0          0   PCI-MSI 354304-edge      PCIe PME, pciehp
 30:          0          0   PCI-MSI 356352-edge      PCIe PME, pciehp
 31:          0          0   PCI-MSI 358400-edge      PCIe PME, pciehp
 32:          0          0   PCI-MSI 360448-edge      PCIe PME, pciehp
 33:          0          0   PCI-MSI 362496-edge      PCIe PME, pciehp
 34:          0          0   PCI-MSI 364544-edge      PCIe PME, pciehp
 35:          0          0   PCI-MSI 366592-edge      PCIe PME, pciehp
 36:          0          0   PCI-MSI 368640-edge      PCIe PME, pciehp
 37:          0          0   PCI-MSI 370688-edge      PCIe PME, pciehp
```

实验箱的中断号

```
linux@localhost:~$ cat   /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5
 14:          0          0          0          0          0          0     GICv3  29 Edge      arch_timer
 15:     190761     140788     155166     125778      23514      15411     GICv3  30 Edge      arch_timer
 17:      31879      14986      14102      10503      57686      10177     GICv3 113 Level     rk_timer
 20:          0          0          0          0          0          0     GICv3  37 Level     ff6d0000.dma-controller
 21:        409          0          0          0          0          0     GICv3  38 Level     ff6d0000.dma-controller
 22:          0          0          0          0          0          0     GICv3  39 Level     ff6e0000.dma-controller
 23:          0          0          0          0          0          0     GICv3  40 Level     ff6e0000.dma-controller
 25:          0          0          0          0          0          0     GICv3  97 Level     dw-mci
 26:      24540          0          0          0          0          0     GICv3  43 Level     mmc1
 27:          0          0          0          0          0          0     GICv3  58 Level     ehci_hcd:usb1
 28:          0          0          0          0          0          0     GICv3  60 Level     ohci_hcd:usb3
 29:          0          0          0          0          0          0     GICv3  62 Level     ehci_hcd:usb2
 30:          0          0          0          0          0          0     GICv3  64 Level     ohci_hcd:usb4
 31:      34567          0          0          0          0          0     GICv3  94 Level     ff100000.saradc
 32:      10386          0          0          0          0          0     GICv3  89 Level     ff3c0000.i2c
 33:      10836          0          0          0          0          0     GICv3  91 Level     ff110000.i2c
 36:          0          0          0          0          0          0     GICv3 129 Level     rockchip_thermal
 38:      51531          0          0          0          0          0     GICv3  88 Level     ff3d0000.i2c
 39:          1          0          0          0          0          0     GICv3  93 Level     rk_pwm_irq
 40:          0          0          0          0          0          0     GICv3 145 Level     ff650000.vpu_service
 41:          0          0          0          0          0          0     GICv3 146 Level     ff650000.vpu_service
 43:          0          0          0          0          0          0     GICv3 148 Level     ff660000.rkvdec
 45:          0          0          0          0          0          0     GICv3  87 Level     rga
 50:          1          0          0          0          0          0     GICv3  51 Level     ff9a0000.gpu
 51:          1          0          0          0          0          0     GICv3  52 Level     ff9a0000.gpu
 52:          1          0          0          0          0          0     GICv3  53 Level     ff9a0000.gpu
 53:          0          0          0          0          0          0     GICv3 151 Level     ff8f0000.vop
 54:       1158          0          0          0          0          0     GICv3 150 Level     ff900000.vop, ff900000.vop
 56:          0          0          0          0          0          0     GICv3  76 Level     rkispl, ff920000.rkispl
```

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

```
linux@localhost:~$ cd /sys
linux@localhost:/sys$ ls
block  class  devices   fs      module  rk8xx
bus    dev    firmware  kernel  power   system_monitor
linux@localhost:/sys$
```

实验箱的“/sys”目录

```
linux@localhost:/sys/devices$ ls
armv8_cortex_a53  breakpoint         platform  system      virtual
armv8_cortex_a72  iio_sysfs_trigger  software  tracepoint
linux@localhost:/sys/devices$
```

实验箱的“/sys/devices”目录

fs3399_linux_example.dts - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

实验箱的设备树文件

/home/linux/worddir/fs3399/system/kernel/arch/arm64/boot/dts/rockchip/fs3399_linux_example.dts

```
/dts-v1/;

#include "fs3399_linux_base.dtsi"

/ {
        compatible = "rockchip,rk3399-fs3399", "rockchip,rk3399";

        leds {
                compatible = "farsight_led";
                led1 = <&gpio4 22 GPIO_ACTIVE_HIGH>;
                led2 = <&gpio0 2 GPIO_ACTIVE_HIGH>;
                led3 = <&gpio0 12 GPIO_ACTIVE_HIGH>;
        };

        farsight_buzzer {
                compatible = "farsight_buzzer";
                gpio1_base = <0xFF730000>;
                pclk_gpio1 = <0xFF750000>;
                status = "okay";
        };

        keys {
                compatible = "farsight_keys";
                key1-gpio = <&gpio1 10 IRQ_TYPE_EDGE_BOTH>;
                key2-gpio = <&gpio0 11 IRQ_TYPE_EDGE_BOTH>;
        };

        rk3399_buzzer {
                compatible = "fs_rk3399_buzzer";
                gpios = <&gpio4 4 GPIO_ACTIVE_HIGH>;
                status = "okay";
        };
```

```
        fs_rk3399_relay {
                compatible = "fs_rk3399_relay";
                gpios = <&gpio2 25 GPIO_ACTIVE_HIGH>;
                status = "okay";
        };

        fs_rk3399_itr {
                compatible = "fs_rk3399_itr";
                gpios = <&gpio1 3 GPIO_ACTIVE_HIGH>;
                status = "okay";
        };

        fs_rk3399_servo{
                compatible = "fs_rk3399_servo";
                gpios = <&gpio4 7 GPIO_ACTIVE_HIGH>;
                status = "okay";
        };
        stepmotor {
                compatible     = "d1-stepmotor";
                        stepmotor_pin1-gpios = <&gpio2 25 GPIO_ACTIVE_LOW>;
                        stepmotor_pin2-gpios = <&gpio1 1 GPIO_ACTIVE_LOW>;
                   stepmotor_pin3-gpios = <&gpio4 7 GPIO_ACTIVE_LOW>;
                stepmotor_pin4-gpios = <&gpio4 4 GPIO_ACTIVE_LOW>;
        };
};

&saradc{
        status = "okay";
};
```

```
&i2c4 {
        status = "okay";

        /*add by mxs 2023/1/3*/
        lm75: lm75@48 {
                compatible = "lm75";
           reg = <0x48>;
                status = "okay";
        };

        zlg7290: zlg7290@38 {
                compatible = "zlg7290";
                reg = <0x38>;
                status = "okay";
        };

        zlg72128: zlg72128@30 {
                compatible = "zlg72128";
           reg = <0x30>;
        status = "okay";
        };

        rfid_13_56: rfid_13_56@19 {
                compatible = "rfid_13_56";
                reg = <0x19>;
        status = "okay";
        };

        rfid_125k: rfid_125k@08 {
        compatible = "rfid_125k";
        reg = <0x08>;
        status = "okay";
        };

        rfid_nfc: rfid_nfc@11 {
        compatible = "rfid_nfc";
        reg = <0x11>;
        status = "okay";
        };
```

```
        rfid_915m: rfid_915m@22 {
        compatible = "rfid_915m";
        reg = <0x22>;
        status = "okay";
        };

        rfid_2_4G: rfid_2_4G@2a {
        compatible = "rfid_2_4G";
        reg = <0x2a>;
        status = "okay";
        };

        dc_motor: dc_motor@4D {
                compatible = "dc_motor";
                reg = <0x4D>;
                dc_motor_pin1-gpios = <&gpio2 25 GPIO_ACTIVE_HIGH>;
                dc_motor_pin2-gpios = <&gpio1 1 GPIO_ACTIVE_HIGH>;
                status = "okay";
         };
};
```

使用设备树文件的驱动程序：基于GPIO子系统的LED灯驱动程序

gpio_leds.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
        return 0;
}

static int farsight_led_resume (struct platform_device *pdev)          //恢复设备文件
{
        return 0;
}

#ifdef CONFIG_OF
static const struct of_device_id led_of_match[] = {                    //匹配表，在of_device_id表中填充成员
    { .compatible = "farsight_led" },
    {}
};
MODULE_DEVICE_TABLE(of, led_of_match);                                 //将设备加入到外设队列，告诉程序员读者，这是个热插拔设备
#endif

//内核中的platform结构体对象，定义了操作对象的方法
static struct platform_driver farsight_led_driver = {
        .probe = farsight_led_probe,                                   //设备树和匹配表匹配成功后之后执行函数，在该函数中，一般初始化设备，申请驱动所需资源
        .remove = farsight_led_remove,                                 //从内核删除设备，释放资源
        .suspend = farsight_led_suspend,                               //挂起
        .resume = farsight_led_resume,                                 //唤醒
        .driver = {                                                    //设备驱动通用属性
                .name = DRIVER_NAME,                                   //设备驱动的名字
                .owner = THIS_MODULE,                                  //表示实现该驱动程序的模块
                .of_match_table = of_match_ptr(led_of_match),          //与设备树进行匹配，使用id_table
        },
};
```

没有使用设备树文件的驱动程序：基于寄存器控制的LED灯驱动程序

reg_leds_driver.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
static int __init farsight_led_dev_init(void)                    //设备文件初始化
{
        int ret_1;
        gpio0_swporta_base = ioremap(0xFF720000,8);                                          //映射GPIO0的基地址
        gpio4_swporta_base = ioremap(0xFF790000,8);                                          //映射GPIO4的基地址
        pclk_gpio0_en = ioremap((0xFF750000+0x104),4);                                       //映射GPIO0的时钟寄存器地址
        pclk_gpio4_en = ioremap((0xFF760000+0x37c),4);                                       //映射GPIO4的时钟寄存器地址
        writel(((readl(pclk_gpio0_en) & ~(0x1<<3)) | 0x1<<19) , pclk_gpio0_en);             //使能GPIO0的时钟
        writel(((readl(pclk_gpio4_en) & ~(0x1<<5)) | 0x1<<21) , pclk_gpio4_en);             //使能GPIO4的时钟
        writel(readl((gpio4_swporta_base + GPIO_DDR)) | 0x1<<22, gpio4_swporta_base + GPIO_DDR);   //设置GPIO4的DDR的相应位为输出模式（LED1）
        writel(readl((gpio0_swporta_base + GPIO_DDR)) | 0x1<<2, gpio0_swporta_base + GPIO_DDR);    //设置GPIO0的DDR的相应位为输出模式（LED2）
        writel(readl((gpio0_swporta_base + GPIO_DDR)) | 0x1<<12, gpio0_swporta_base + GPIO_DDR);   //设置GPIO0的DDR的相应位为输出模式（LED3）
        cdev = kmalloc(sizeof(struct cdev), GFP_KERNEL);
        cdev_init(cdev,&farsight_led_ops);                                   //初始化字符设备对象
        cdev->owner = THIS_MODULE;                                           //设置字符设备所属模块
        ret_1 = alloc_chrdev_region(&devnum,0,1,"leds_reg_device");          //申请设备号
        if(ret_1<0)
        {
                goto out_err_0;
        }
        ret_1 = cdev_add(cdev, devnum, 1);                                   //添加字符设备
        if (ret_1 < 0)
        {
                goto out_err_1;
        }
        class = class_create(THIS_MODULE,"leds_reg_class");                  //创建character类
        if (IS_ERR(class))
        {
                ret_1 = PTR_ERR(class);
                goto out_err_2;
        }
        dev = device_create(class,NULL,devnum,NULL,"leds_reg_device");       //创建字符设备节点
        if (IS_ERR(dev))
        {
                ret_1 = PTR_ERR(dev);
                goto out_err_3;
        }
        return 0;
```

• 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

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

```
linux@localhost:/sys/bus$ ls
amba          cpu           mdio_bus  pci          snd_seq
cec           event_source  media     pci_express  spi
clockevents   hid           mipi-dsi  platform     usb
clocksource   i2c           mmc       scsi         usb-serial
container     iio           nvmem     sdio         workqueue
linux@localhost:/sys/bus$
```

实验箱的“/sys/bus”目录

```
linux@localhost:/sys/class$ ls
android_usb     extcon       misc          rfkill        thermal
ata_device      gpio         mmc_host      rkvdec        tpm
ata_link        graphics     net           rkwifi        tty
ata_port        hidraw       nvme          rtc           udc
backlight       hwmon        pci_bus       scsi_device   usbmon
bdi             i2c-adapter  phy           scsi_disk     vc
block           i2c-dev      power_supply  scsi_host     video4linux
bluetooth       ieee80211    ppp           sound         vpu_service
bsg             input        pps           spi_host      vtconsole
devcoredump     iommu        ptp           spi_master    zlg72xx-old
devfreq         leds         pwm           spi_transport zram-control
devfreq-event   leds_ctl     rc            spidev
dma             mdio_bus     regulator     stepmotor
drm             mem          rfid          switch
linux@localhost:/sys/class$
```

实验箱的“/sys/class”目录

```
linux@localhost:/sys/bus/usb/drivers$ ls
asix          cdc_mbim  keyspan   rndis_host     uas          usbserial_generic
ax88179_178a  cdc_ncm   option    rndis_wlan     usb          usbtouchscreen
bfusb         cdc_wdm   oti6858   rtl8150        usb-storage  uvcvideo
btusb         cp210x    pl2303    rtl8821cu      usbfs
cdc_acm       ftdi_sio  qcserial  sierra         usbhid
cdc_ether     hub       r8152     snd-usb-audio  usbserial
linux@localhost:/sys/bus/usb/drivers$
```

实验箱的“/sys/bus/usb/drivers”目录

• 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

```
linux@localhost:/sys/bus$ ls
amba          cpu           mdio_bus  pci          snd_seq
cec           event_source  media     pci_express  spi
clockevents   hid           mipi-dsi  platform     usb
clocksource   i2c           mmc       scsi         usb-serial
container     iio           nvmem     sdio         workqueue
linux@localhost:/sys/bus$
```

```
linux@localhost:/sys/bus/platform$ ls
devices  drivers  drivers_autoprobe  drivers_probe  uevent
linux@localhost:/sys/bus/platform$
```

实验箱的“/sys/bus/platform”目录

gpio_leds.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
        return 0;
}

static int farsight_led_resume (struct platform_device *pdev)
{
        return 0;
}

#ifdef CONFIG_OF
static const struct of_device_id led_of_match[] = {
    { .compatible = "farsight_led" },
    {}
};
MODULE_DEVICE_TABLE(of, led_of_match);
#endif

//内核中的platform结构体对象，定义了操作对象的方法
static struct platform_driver farsight_led_driver = {
        .probe = farsight_led_probe,
        .remove = farsight_led_remove,
        .suspend = farsight_led_suspend,
        .resume = farsight_led_resume,
        .driver = {
                .name = DRIVER_NAME,
                .owner = THIS_MODULE,
                .of_match_table = of_match_ptr(led_of_match),
        },
};
```

基于GPIO子系统的LED灯驱动程序的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

在实验箱的“Xshell”上执行：cat /proc/iomem

```
root@localhost:/sys/bus/platform# cat /proc/iomem
00110000-0014ffff : persistent_ram
00150000-001cffff : persistent_ram
00200000-083fffff : System RAM
  00280000-010affff : Kernel code
  011c0000-0151afff : Kernel data
0a200000-7fffffff : System RAM
fe300000-fe30ffff : /ethernet@fe300000
fe320000-fe323fff : /dwmmc@fe320000
fe330000-fe33ffff : mmc1
```

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

- 阅读第4次实验的设备驱动程序的源代码

实验1：内核模块
实验2：字符设备驱动
实验3：3个LED灯（基于寄存器控制）
实验4：3个LED灯（基于GPIO子系统控制）
实验5：呼吸灯（LED1灯）
实验6：蜂鸣器
实验7：2个按键
实验8：温度传感器（I2C总线）
实验9：小键盘/数码管
实验10：ADC信号采集（电位器+4个传感器）
实验11：继电器
实验12：光电开关
实验13：蜂鸣器（实验箱底板）
实验14：陀机
实验15：步进电机
实验16：直流电机

Thanks