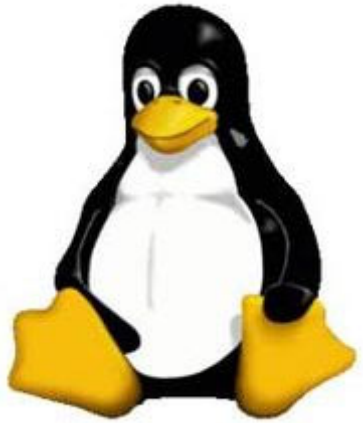
华清远见
FARSIGHT®
始于2004
嵌入式培训专家

嵌入式开发之linux驱动开发入门

易老师，深圳中心

专业始于专注　　卓识源于远见



Topics...

1.LINUX驱动干什么用的？

2.现在企业需要什么底层驱动人才？

3.如何成为嵌入式底层人才？

4.LINUX驱动和单片机开发有何异同

5.LINUX设备驱动的架构分析

6.LCD的硬件原理分析

7.LCD的帧缓冲框架分析

8.LCD的应用编程方法

9.LCD框架实现

10..LINUX的平台设备机制分析

11.Linux设备驱动之LCD驱动开发实例分析

12.LINUX设备驱动开发入门总结

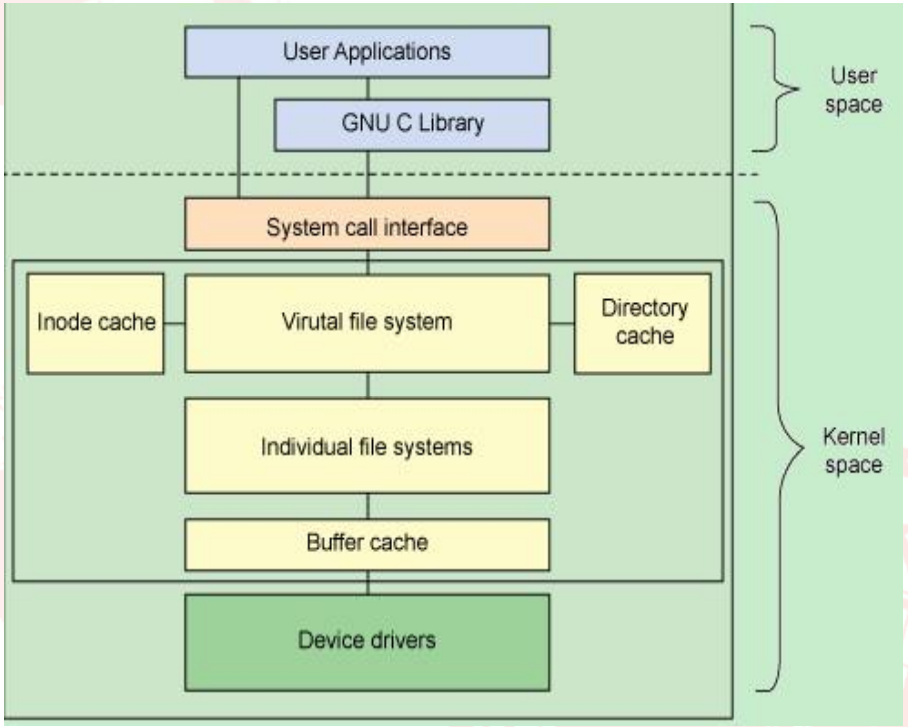
13.华清对驱动课程设置

1.LINUX驱动作用

1.管理和操作硬件

2.为上层应用操作各种硬件提供统一的"武器"（调用接口）

LINUX组成结构图



2.企业对底层驱动人才的基本需求

1.熟悉C语言,有良好的代码习惯
2.熟悉Linux设备驱动软件架构
3.熟悉XXX驱动（或有XXX驱动开发经验）
4.有XXXXX等开发经验者优先考虑。
5.具有强烈的责任心，良好的沟通能力及团队合作精神。
6.有X年以上驱动开发经验

3.如何成为嵌入式底层人才

方式1: 参与企业项目开发

方式2：集中培训学习

1.通过集中培训系统掌握LINUX驱动框架

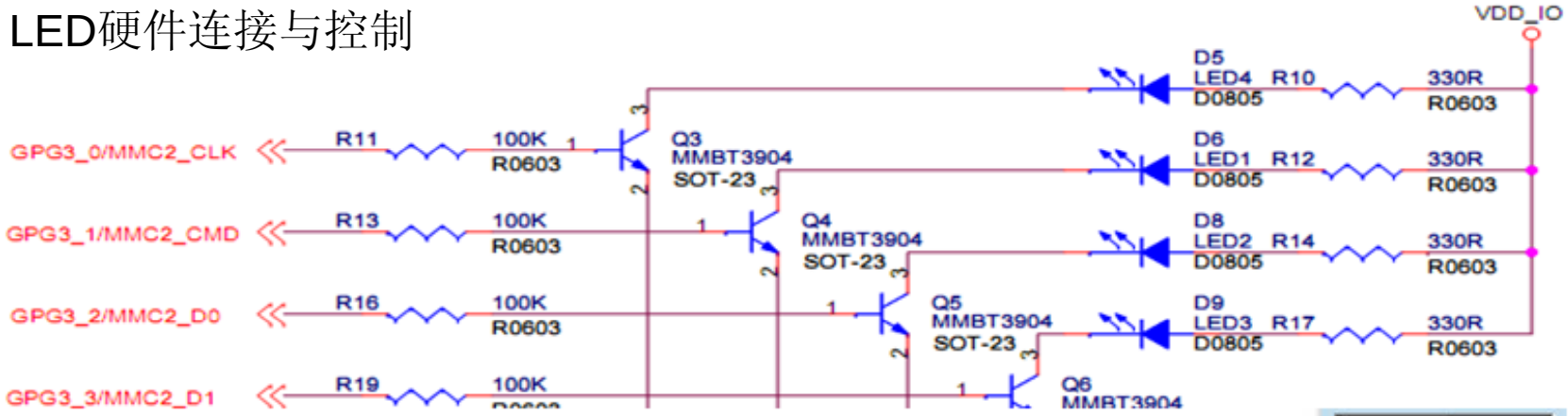
2.通过培训中的实战项目培养动手能力，
并加深对驱动框架原理性理解

3.积累硬件工作原理和调试经验

4.平时多总结、揣摩、提炼

4.单片机开发和LINUX驱动开发的异同

LED硬件连接与控制



Register	Address	R/W	Description	Reset Value
			up/down Register	
GPG3CON	0xE030_01C0	R/W	Port Group GPG3 Configuration Register	0x00000000
GPG3DAT	0xE030_01C4	R/W	Port Group GPG3 Data Register	-
GPG3PUD	0xE030_01C8	R/W	Port Group GPG3 Pull-up/down Register	0x1555
GPG3DRV	0xE030_01CC	R/W	Port Group GPG3 Drive strength control Register	0x0000
GPG3PDNCON	0xE030_01D0	R/W	Port Group GPG3 Power down mode Configuration Register	0x00
GPG3PDNPULL	0xE030_01D4	R/W	Port Group GPG3 Power down mode Pull-up/down Register	0x00

4.单片机开发和LINUX驱动开发的异同(cont.)

5.1.15 Port Group GPG3 Configuration Register (GPG3CON, R/W, Address = 0xE030_01C0)

Field	Bit	Description	Reset Value
GPG3CON[0]	[3:0]	0000 = Input, 0001 = Output, 0010 = SD_2_CLK, 0011 = SPI_2_CLK, 0100 = I2S2_SCLK, 0101 = PCM_0_SCLK, 1111 = NWU_INTG14[0]	0000
GPG3CON[1]	[7:4]	0000 = Input, 0001 = Output, 0010 = SD_2_CMD, 0011 = SPI_2_nSS, 0100 = I2S2_CDCLK, 0101 = PCM_0_EXTCLK, 1111 = NWU_INTG14[1]	0000

编程思路：

GPG3（0-3）控制LED亮灭

当各位输出高电平时LED形成通路，此时LED亮，反之灭。

对GPG3CON进行操作，将该4位设成输出位

对GPG3DAT进行移位写值操作，控制各LED亮灭。

4.单片机开发和LINUX驱动开发的异同(cont.)

boot.S

```
//系统引导代码[汇编]
略
//跳转到C代码
 bl main
```

C点LED灯(跑马灯)

Led_loop.c

```
#define    GPG3CON       (*(volatile unsigned int *)0xE03001C0)
#define    GPG3DAT       (*(volatile unsigned int *)0xE03001C4)
#define    MY_DELAY      0x18000
int main(void)
{
 GPG3CON &|= 0xffff;
  GPG3CON  |= 0x1111;
  unsigned int i;
```

```
  for ( i = 0; i < MY_DELAY*10; i++);
  GPG3DAT |= 0x0; //clear all LEDS
while (1) {

    GPG3DAT |=(1<<0);
    for ( i = 0; i < MY_DELAY; i++);
    GPG3DAT &=~(1<<0);
    GPG3DAT |= (1<<1);

    for ( i = 0; i < MY_DELAY; i++);
    GPG3DAT |= (1<<2);
    GPG3DAT &=~(1<<1);
    for ( i = 0; i < MY_DELAY; i++);
    GPG3DAT |= (1<<3);
    GPG3DAT &=~(1<<2);

    for ( i = 0; i < MY_DELAY; i++);
    GPG3DAT &=~0x0;   //clear all LEDS
  }
  return 0;
}
```

4.单片机开发和LINUX驱动开发的异同(cont.)

Linux点灯

```
//应用层

dev_fd =  open("/dev/leds, …);

while(1)
      {
      ioctl(dev_fd,LED_ON,0);  //点灯
      sleep(1); //延迟
      ioctl(dev_fd,LED_OFF,0); //灭灯
      sleep(1); //延迟
      }
close(dev_fd);
```

```
//驱动

注册LED字符设备和LED操作函数
 register_chrdev(major, "led", &led_fops);

//关键结构
static struct file_operations leds_remap_ops = {
       .owner   = THIS_MODULE,
       .open    = leds_open,   //open
       .release = leds_release,
      .ioctl   = leds_ioctl,
};
///点灯实现
static int leds_ioctl(…, unsigned int cmd, …)
{
       switch ( cmd )
       {
              case LED_ON:
                     {
                            led_on();  //具体硬件操作
                            break;
                     }
              case LED_OFF:
                     {
                            led_off(); //具体硬件操作

                            break;
                     }
             //…
     }
     return 0;
}
```

4.LINUX驱动开发入门小结

LINUX的语录：

1.linux驱动为应用操作硬件提供统一的接口

2.LINUX系统中把任务一分为二，应用制定策略，驱动实现机制

--选自《StephenYee, farsight Inc.》

让我们理解一下：

a.应用制定策略---要做什么---想法问题

驱动实现机制----能做什么---能力问题

b. 军官与士兵的关系

应用----军官

驱动----士兵

5.LINUX驱动框架分析

LINUX驱动在内核中的位置和分类

1.是内核中的一部分

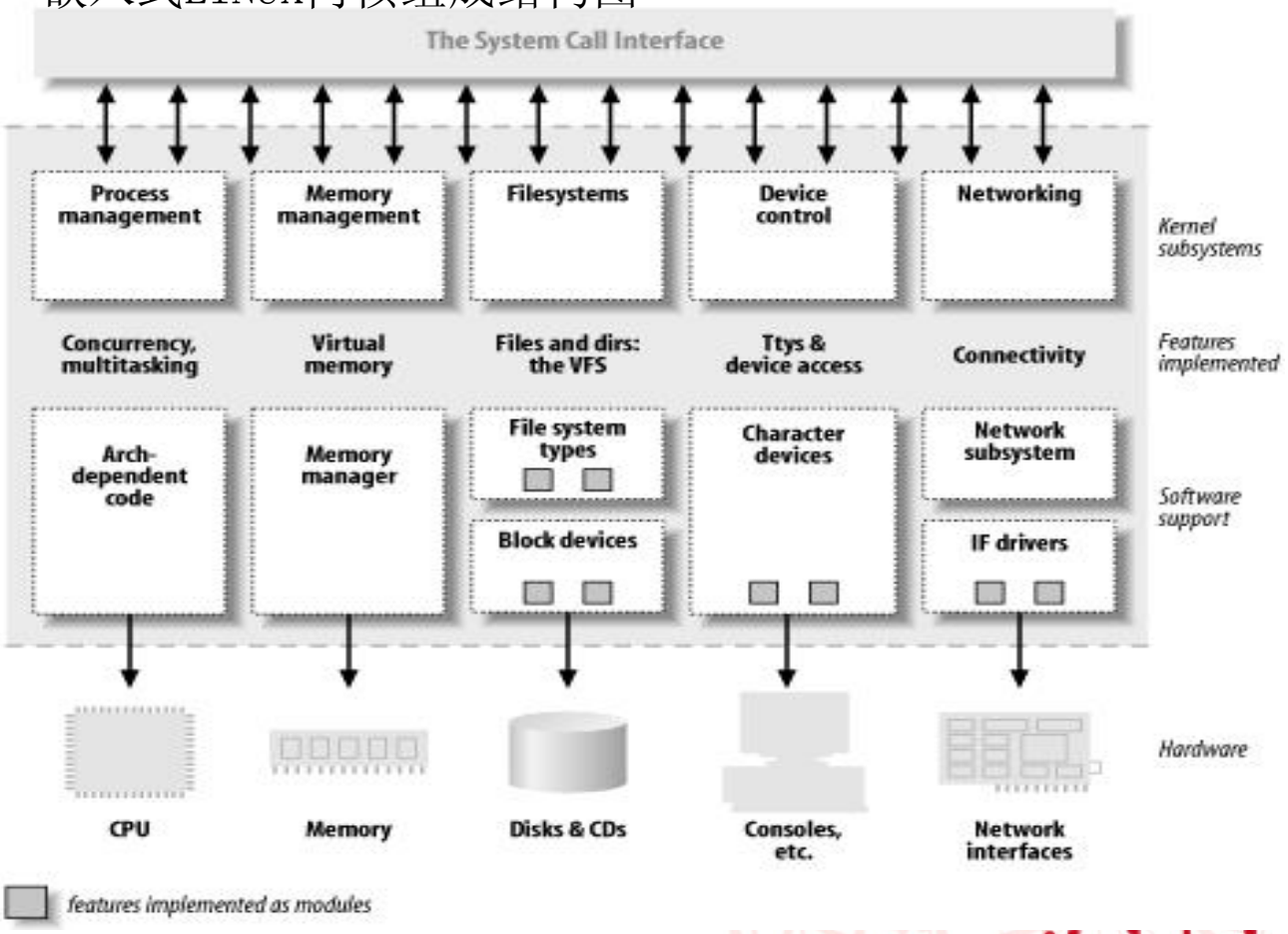
2.分为三类设备

字符设备

块设备

网络设备

嵌入式LINUX内核组成结构图



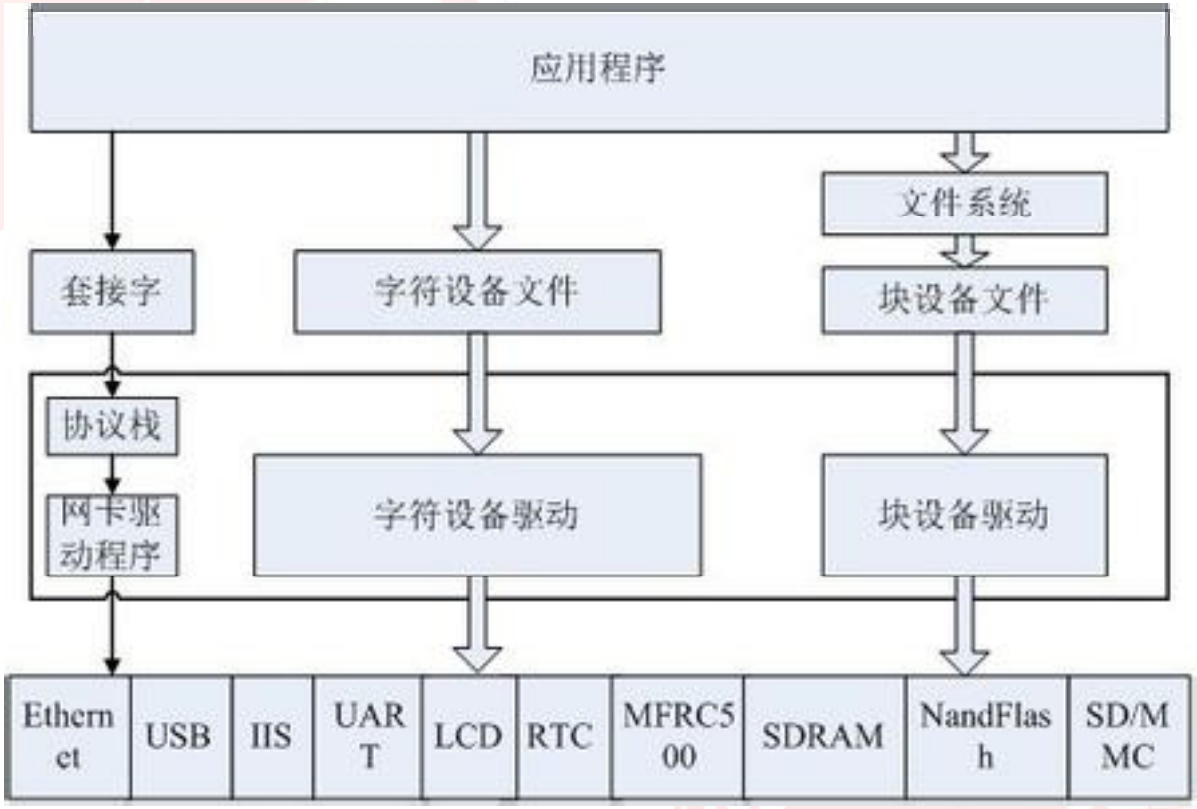
5.LINUX驱动框架分析(cont)

---LINUX三类设备实例

字符设备
块设备
网络设备

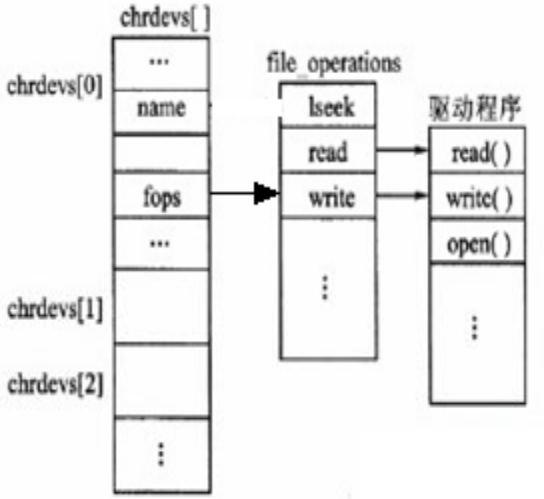
Linux内核中：
用C语言实现的面向对象编程

LINUX设备分类



5.LINUX驱动框架分析(Cont.)

Linux内核中：
用C语言实现的面向对象编程

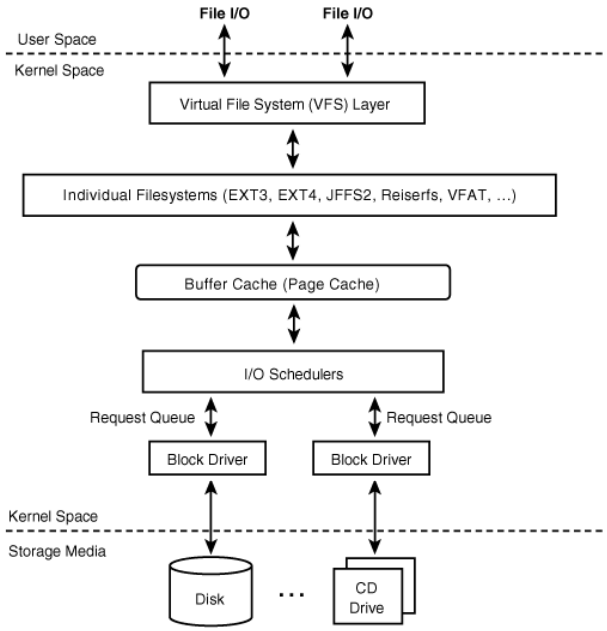


- **字符设备的file_operations**

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
       ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        ……
};
```

5.LINUX驱动框架分析(Cont.)

Linux内核中：
用C语言实现的面向对象编程

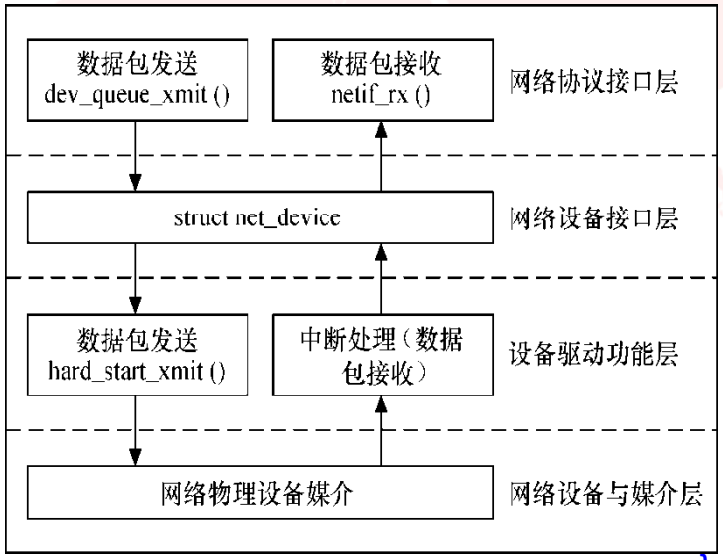


块设备的block_device_operations

```
struct block_device_operations {
        int (*open) (struct block_device *, fmode_t);    //打开
        int (*release) (struct gendisk *, fmode_t);   //释放
        int (*ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);//ioctl命 令控制
        int (*media_changed) (struct gendisk *); //介质是否发生改变
        int (*revalidate_disk) (struct gendisk *);//重新使能磁盘有效
        int (*getgeo)(struct block_device *, struct hd_geometry *);//获取磁盘几何信息
        struct module *owner;
        ……
};
```

5.LINUX驱动框架分析(Cont.)

Linux内核中：
用C语言实现的面向对象编程

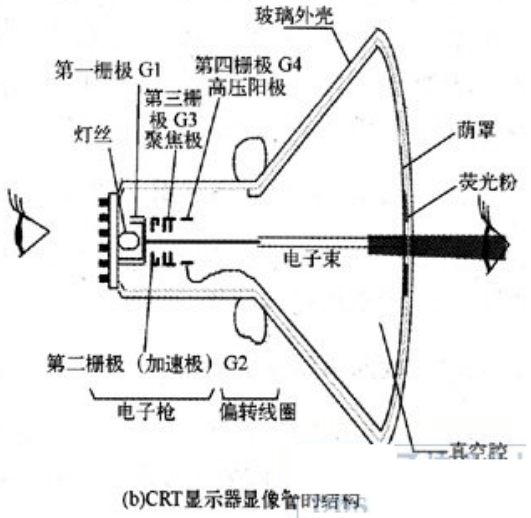


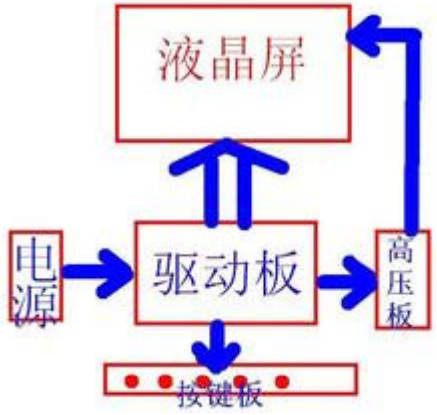
网络设备的net_device

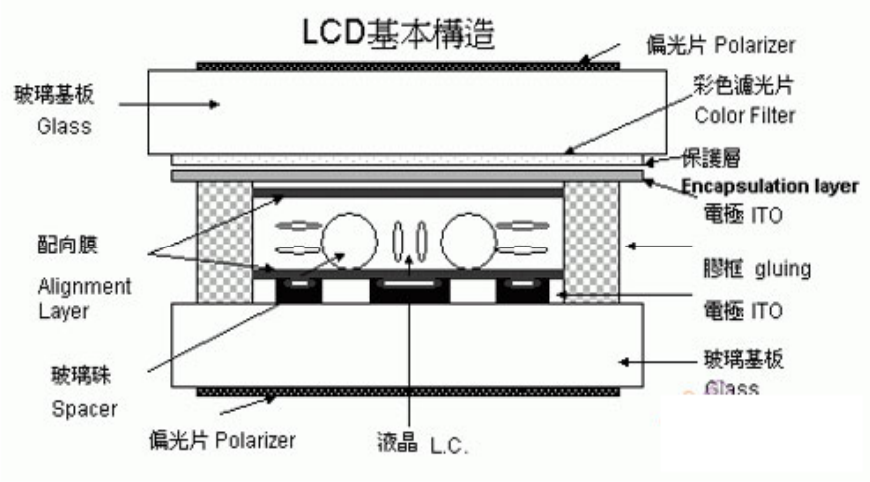
```
struct net_device {
    char                        name[IFNAMSIZ];
    unsigned long               mem_end;/* shared mem end    */
    unsigned long               mem_start;/* shared mem start  */
    unsigned long               base_addr;/* device I/O address */
    unsigned int                irq;          /* device IRQ number */
    struct net_device_stats     stats;
    const struct net_device_ops *netdev_ops;
    unsigned char               *dev_addr;/* hw address */
    ....//此处省略350行...
};
```

6.6.LCD硬件原理分析(cont.)

- 按显示设备所用的显示器件分类，有阴极射线管(CRT)显示器、液晶显示器(LCD)、等离子显示器等。
- 按所显示的信息内容分类，有字符显示器、图形显示器、图像显示器三大类。
- 分辨率：是指显示器所能表示的像素个数。如1024*768，800*600等。
- 灰度级：是指黑白显示器中所显示的像素点的亮暗差别，在彩色显示器中则表现为颜色的不同。







帧缓冲(Framebuffer)显示技术

- 帧缓冲(framebuffer）提供给图形应用程序一个定义良好的接口，隐藏了底层硬件的细节。图形应用程序无需了解硬件寄存器，时序等信息。

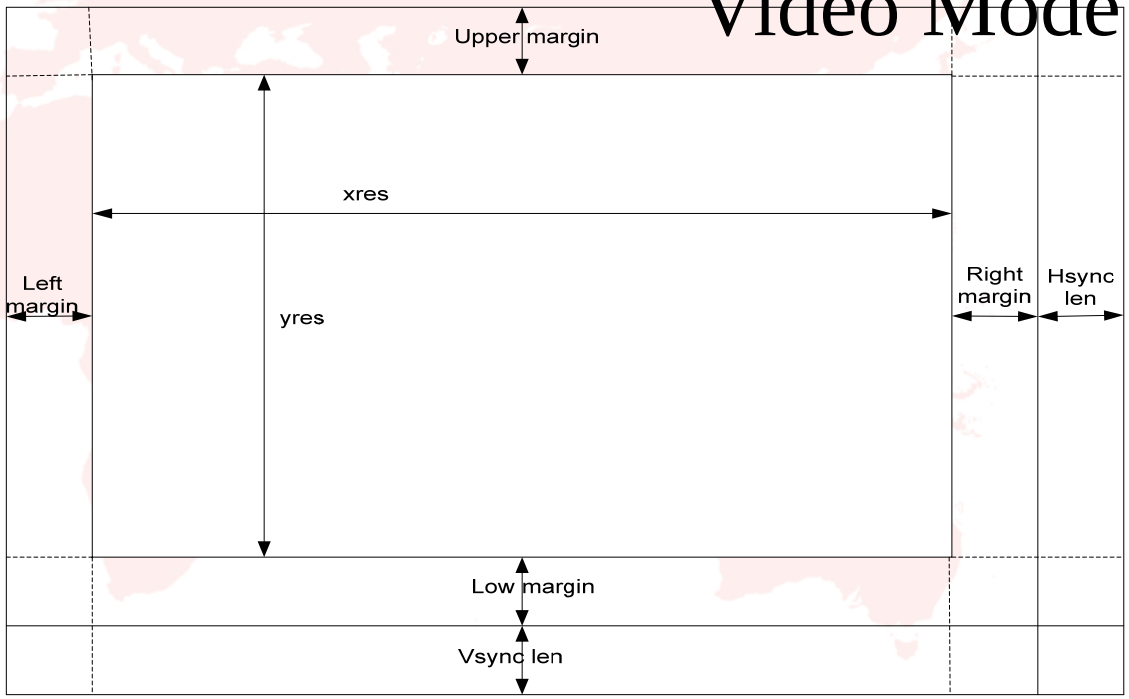
<table>
<tr><td colspan="2">Application code</td></tr>
<tr><td colspan="2">GUI Lib</td></tr>
<tr><td>Linux Kernel</td><td>Frambuffer</td></tr>
<tr><td colspan="2">Hard ware</td></tr>
</table>

6.帧缓冲(Framebuffer)显示技术(cont.)

- Framebuffer支持5种颜色显示方式：
  - 单色（Monochrome）
  - 伪彩色（Pseudo color）
  - 真彩色（True color）
  - 直接彩色（Direct color）
  - 灰度（Grayscale displays）

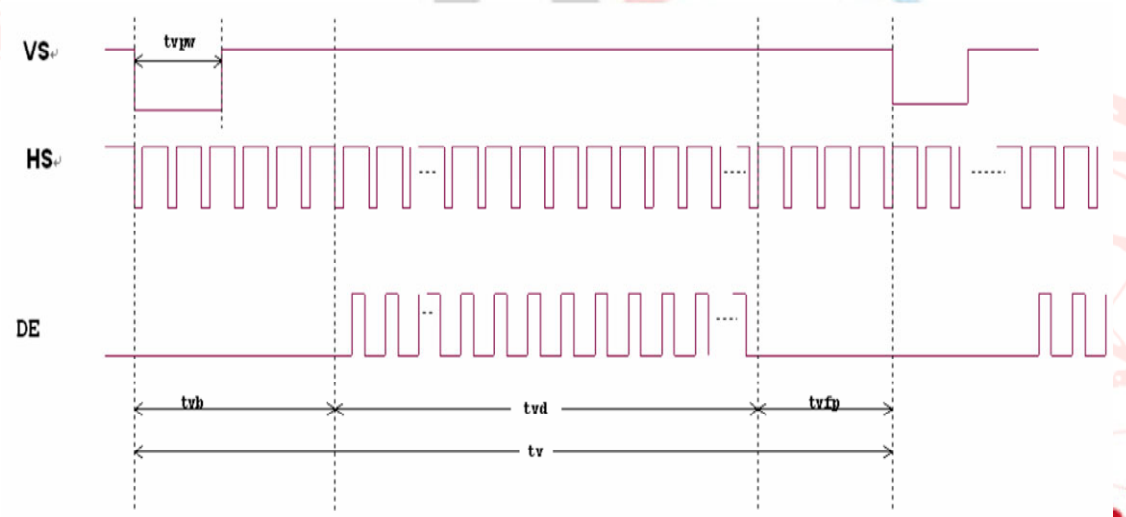
6.帧缓冲(Framebuffer)显示技术(cont.)

Video Mode timings



6.帧缓冲(Framebuffer)显示技术

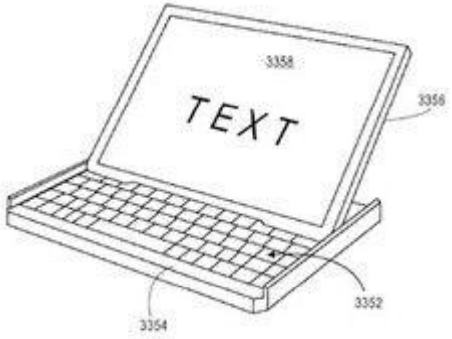
FS210-LCD时序



7.LCD的应用编程方法

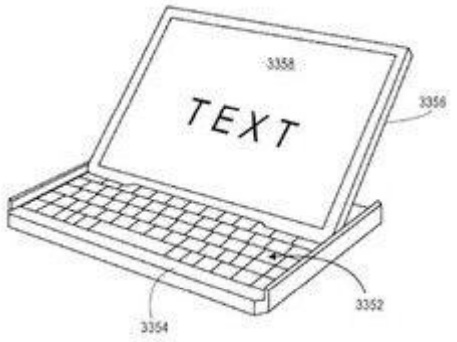
- Framebuffer为应用程序提供了良好的接口；
- Framebuffer驱动程序可以作为普通memory类型的设备，用户可以直接读写设备的内容；
- 比如cp /dev/fb0 screenshot，即是把fb0的内容拷贝到截图文件中

0 = /dev/fb0 First frame buffer
1 = /dev/fb1 Second frame buffer
...
31 = /dev/fb31 32nd frame buffer



7.LCD的应用编程方法(Cont.)

- Framebuffer驱动程序实现上，类似于/dev/mem设备；
- 它支持read,write,seek和mmap。mmap是使用Framebuffer的主要方式；
- Framebuffer使用显示设备中的帧内存，区别于/dev/mem设备使用这个主内存；
- 另外Framebuffer支持一些特定的ioctl，主要用来获得设备信息和设置设备参数。比如我们访问调色板就是通过ioctl方法实现。



8.LCD的帧缓冲框架分析

- 在内核中，帧缓冲设备可以工作于模块中，允许动态加载。这类驱动必须调用register_framebuffer在系统中注册；
- Framebuffer 只是一个提供显示内存和显示芯片寄存器从物理内存映射到进程地址空间中的设备；
- 对于应用程序而言，如果希望在 FrameBuffer 之上进行图形编程，还需要自己动手完成其他许多工作。

8.LCD的帧缓冲框架分析(Cont.)

- 在应用程序中，操作帧缓冲设备（/dev/fb）的一般步骤如下：
  - 打开/dev/fb设备文件。
  - 用ioctl操作取得当前显示屏幕的参数。
  - 将屏幕缓冲区映射到用户空间。
  - 映射后就可以直接读写屏幕缓冲区，进行绘图和图片显示了。

9.LCD框架实现-关键数据结构(Cont.)

- fb_fix_screeninfo
- fb_var_screeninfo
- fb_cmap
- fb_info
- fb_ops

9.LCD框架实现-关键数据结构(Cont.)

fb_fix_screeninfo结构(1)

```
struct fb_fix_screeninfo {
   unsigned long smem_start;
__u32 smem_len;
__u32 type;
__u32 visual;
   ……
};
```

- 该结构用来描述设备无关，不可变更的信息。用户可以使用FBIOGET_FSCREENINFO命令来获得这些信息

9.LCD框架实现-关键数据结构(Cont.)

fb_fix_screeninfo结构(2)

结构包含主要的项

- unsigned long smem_start;描述缓冲区起始地址（物理地址）
- u32 smem_len;描述缓冲区长度
- u32 type;描述fb类型，比如TFT或STN类型
- u32 visual;描述显示颜色是真彩色，伪彩色还是单色

9.LCD框架实现-关键数据结构(Cont.)

fb_var_screeninfo结构(1)

```
struct fb_var_screeninfo{
    __u32 xres;
    __u32 yres;
    __u32 xres_virtual;
    __u32 yres_virtual;
    __u32 xoffset;
    __u32 yoffset;
    ……
}
```

- 该结构描述设备无关的，可更改的配置信息。应用程序可以使用FBIOGET_VSCREENINFO命令获得这些信息，使用FBIOPUT_VSCREENINFO命令写入这些信息

9.LCD框架实现-关键数据结构(Cont.)

fb_var_screeninfo结构(2)

主要的数据项包含

- __u32 xres; //可见分辨率
- __u32 yres;
- __u32 xres_virtual; // 虚拟分辨率
- __u32 yres_virtual;
- __u32 xoffset; //从虚拟到可见分辨率的偏移
- __u32 yoffset;
- 以及屏幕四周的margin，像素时钟，同步等时序信息

9.LCD框架实现-关键数据结构(Cont.)

fb_cmap结构(1)

```
struct fb_cmap {
    __u32 start;
    __u32 len;
    __u16 *red;
    __u16 *green;
    __u16 *blue;
    __u16 *transp;
};
```

- 设备无关的调色板结构。用来描述调色板中的颜色

9.LCD框架实现-关键数据结构(Cont.)

fb_cmap结构(2)

主要数据项包括

- __u32 start;描述调色板起始位置
- __u32 len;描述总共颜色数
- __u16 *red;红色的值
- __u16 *green;绿色
- __u16 *blue;蓝色
- __u16 *transp;透明度

9.LCD框架实现-关键数据结构(Cont.)

fb_info结构(1)

```
struct fb_info{
    struct fb_var_screeninfo var;
    struct fb_fix_screeninfo fix;
    struct fb_monspecs monspecs;
    struct fb_cmap cmap;
    struct fb_ops *fbops;
    ……
}
```

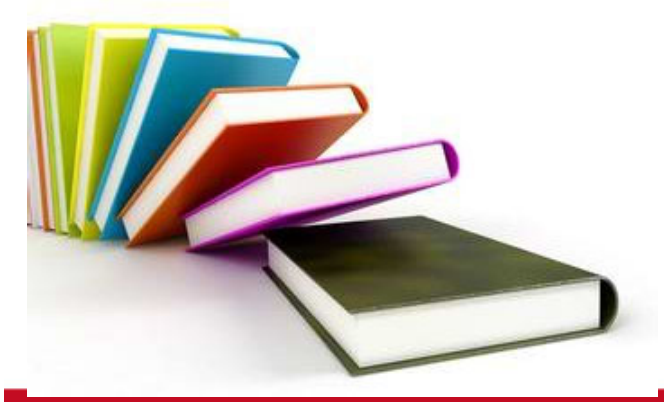
- fb_info结构是Linux为帧缓冲设备定义的驱动层接口，用于用户在内核空间的调用。它不仅包含了底层函数，而且还有记录设备状态的数据。每个帧缓冲设备都有一个fb_info结构相对应。

9.LCD框架实现-关键数据结构(Cont.)

fb_info结构(2)

主要数据项包括

- struct fb_var_screeninfo var;描述当前的可变参数
- struct fb_fix_screeninfo fix;描述当前的固定参数
- struct fb_monspecs monspecs;描述当前显示器的特有固定信息
- struct fb_cmap cmap;描述当前颜色映射表
- struct fb_ops *fbops;指向驱动设备工作所需的函数集，fb_ops 用户应用可以使用ioctl()系统调用来操作设备



10.Linux平台设备机制分析

bus(总线)

系统中总线由struct bus_type描述，定义为：

```
struct bus_type {
    char * name; 总线类型的名称
    struct subsystem subsys; 与该总线相关的subsystem
    struct kset drivers; 所有与该总线相关的驱动程序集合
    struct kset devices; 所有挂接在该总线上的设备集合
    struct bus attribute * bus_attrs; 总线属性
    struct device attribute * dev_attrs; 设备属性
    struct driver attribute * drv_attrs; 驱动程序属性
    int (*match)(struct device * dev, struct device_driver * drv);
    int (*hotplug) (struct device *dev, char **envp, int num_envp,        char *buffer, int
    buffer_size);
    int (*suspend)(struct device * dev, u32 state);
    int (*resume)(struct device * dev);
};
```

10.Linux平台设备机制分析(Cont.)

device(设备)

系统中的任一设备在设备模型中都由一个device对象描述，其对应的数据结构 struct device定义为：

```
struct device {
struct list_head g_list;
struct list_head node;
struct list_head bus_list;//将连接到相同总线上的设备组织成链表
struct list_head driver_list; //将同一驱动程序管理的所有设备组织为链表
struct list_head children;
struct device *parent;
struct kobject kobj;//用于引用计数管理并通过它实现设备层次结构
char bus_id[BUS_ID_SIZE];
struct bus_type *bus;//域描述设备所连接的总线类型
struct device_driver *driver;//指向管理该设备的驱动程序对象
void *driver_data;
/* Several fields omitted */
};
```

10.Linux平台设备机制分析(Cont.)

driver(驱动)

系统中的每个驱动程序由一个device_driver对象描述，对应的数据结构定义为：

- struct device_driver {
  char *name; 设备驱动程序的名称
  struct bus_type *bus; 该驱动所管理的设备挂接的总线类型
  struct kobject kobj; 内嵌kobject对象
  struct list_head devices; 该驱动所管理的设备链表头
  int (*probe)(struct device *dev); 指向设备探测函数，用于探测设备是否可以被该驱动程序管理
  int (*remove)(struct device *dev); 用于删除设备的函数
  /* some fields omitted*/
};

10.Linux平台设备机制分析(Cont.)

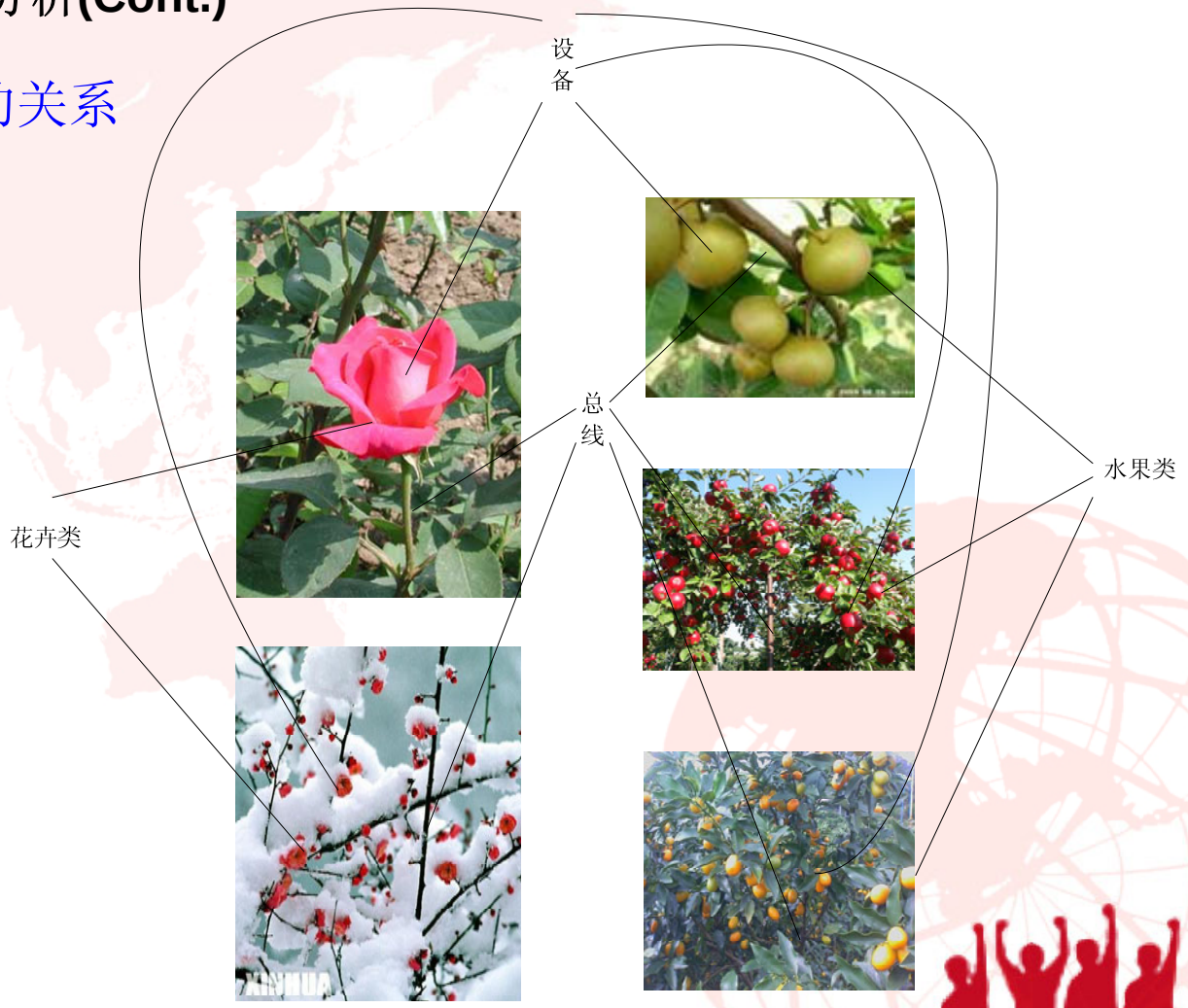
class(类)

系统中的设备类由struct class描述，表示某一类设备。所有的class对象都属于class_subsys子系统，对应于sysfs文件系统中的/sys/class目录

```
struct class
{
        const char                      * name;  //类名
        struct module                   * owner;
        struct subsystem        subsys; //对应的subsystem
        struct list_head        children; //class_device链表
        struct list_head        interfaces; //class_interface链表
        struct semaphore        sem;            //children和interfaces链表锁
        struct class_attribute                          * class_attrs;//类属性
        struct class_device_attribute       * class_dev_attrs;//类设备属性
        int         (*uevent)(struct class_device *dev, char **envp,
    int num_envp, char *buffer, int buffer_size);//事件
    void        (*release)(struct class_device *dev);
        void        (*class_release)(struct class *class);
};
```

10.Linux平台设备机制分析(Cont.)

设备、总线与类之间的关系



11.Linux设备驱动之LCD驱动开发实例分析

基于CORTEX-A8平台的LCD屏驱动代码分析

4.LINUX驱动开发入门总结

1.理解inux驱动将具体任务一分为二、面向对象的设计思想

2.理解LINUX中 应用制定策略、驱动实现机制的基本原则

3.具体到具体驱动的时候以关键的数据结构为纲，深入掌握1,2个驱动，按照通1,2个驱动积累的经验划再分门别类进行学习

4. 原理联系实际，多动手调试；实际回归印证原理

12.LINUX驱动开发入门总结

1.理解inux驱动将具体任务一分为二、面向对象的设计思想

2.理解LINUX中 应用制定策略、驱动实现机制的基本原则

3.具体到具体驱动的时候以关键的数据结构为纲，深入掌握1,2个驱动，按照通1,2个驱动积累的经验划再分门别类进行学习

4. 原理联系实际，多动手调试；实际回归印证原理

13.华清驱动课程设置（1）

第一天：
1.linux设备驱动开发基础知识
2.Linux内核模块开发
3.Linux字符设备驱动结构讲解

第二天：
1.Linux设备驱动之的并发机制的实现
2.Linux设备驱动之阻塞与非阻塞IO的实现
3.Linux设备驱动之POLL及select机制
4.Linux设备驱动之异步通知机制的实现

第三天：
1.Linux设备驱动之中断编程
2.Linux设备驱动之中断底半部机制
3.Linux设备驱动之内核定时器

第四天：
1.Linux设备驱动之内存管理理
2.Linux设备驱动之设备驱动模型
3.Linux设备驱动之平台设备驱动机制

13.华清课程设置（2）

第五天：

1.Linux设备驱动之GPIO接口驱动编程
2.Linux设备驱动之按键中断接口驱动编程
3.Linux设备驱动之看门狗接口驱动编程
4.Linux设备驱动之PWM接口驱动编程

第六天：

1.Linux设备驱动之ADC接口驱动编程
2.Linux设备驱动之I2C接口驱动编程

第七天：

1.Linux设备驱动之SPI接口驱动编程
2.Linux设备驱动之块设备驱动开发

第八天：

linux设备驱动之网卡驱动开发