# 总结

|  |
| --- |
| Fb子系统  register\_framebuffer(struct fb\_info \*fb\_info) //注册fb[i]设备  unregister\_framebuffer(struct fb\_info \*fb\_info) //注销fb[i]设备  owl\_display子系统：  1个Display Engine有4个macroLayer层  1个MarcoLayer有4个sublayer层，可以相互叠加  **owl\_display\_device\_register**(name,dev,devdata,display\_id,modes,num\_modes,ops);  **owl\_display\_device\_unregister**(struct owl\_display\_device \*dev);  //注册/注销 显示设备到/sys/class/owl\_display子系统中  struct owl\_display\_device \***get\_owl\_display**(int display\_id);  struct owl\_display\_device \***owl\_display\_connect**(int display\_id);  **owl\_display\_disconnect**(struct owl\_display\_device \*disp\_dev);  //通过display\_id在display\_list链表中找到对应的owl\_display\_device显示设备  //connect()会调用owl\_display\_device ->ops->update\_status(owl\_display\_device); |
|  |

# 1.1 Fb设备框架的构建: fb\_info

kernel\drivers\video\fbmem.c

subsys\_initcall(fbmem\_init);

## fbmem\_init() : 构建fb子系统

1. 构建字符设备: **register\_chrdev(29, “fb”, ops)**

2. 构建FB设备框架 : fb\_class

|  |
| --- |
| **struct** **class** \*fb\_class; |
| fbmem\_init(void)  1. 创建/proc/fb文件  //proc\_create(name, mode, parent, proc\_fops)  |--**proc\_create**("fb", 0, NULL, &**fb\_proc\_fops**);  2. 创建注册字符设备: fb: 29  //#define FB\_MAJOR 29 /\* /dev/fb\* framebuffers \*/  |--**register\_chrdev**(FB\_MAJOR,**"fb"**,&**fb**\_**fops**);   |  | | --- | | static const **struct** **file**\_**operations** **fb**\_**fops** = {  .owner = THIS\_MODULE,  .read = fb\_read,  .write = fb\_write,  .unlocked\_ioctl = fb\_ioctl,  .compat\_ioctl = fb\_compat\_ioctl,  .mmap = fb\_mmap,  .open = fb\_open,  .release = fb\_release,  .get\_unmapped\_area = get\_fb\_unmapped\_area,  .fsync = fb\_deferred\_io\_fsync,  .llseek = default\_llseek,  } |   3. 创建fb的class系统: /sys/class/graphics  |--**fb**\_**class** = **class\_create**(THIS\_MODULE, "**graphics**"); |

## register\_framebuffer(fb\_info \*) : 注册fb\_info到fb\_class中

kernel\drivers\video\fbmem.c

1. 构建注册fb设备的必备参数: fb\_info

2. 在fb[0-31]的范围内寻找可用的node id, 将fb\_info挂载到对应的**registered\_fb[i]**中

3. 创建/dev/fb[i]设备 ： **device\_create(fb\_class, fb\_info->device, MKDEV(29, i), “fb[i]”);**

|  |
| --- |
| 调用关系:  register\_framebuffer(fbdev->fbi);  owlfb\_create\_framebuffer()  owlfb\_probe() |

|  |
| --- |
| **struct fb\_info** \***registered\_fb**[FB\_MAX];  #define FB\_MAX 32 /\* sufficient for now \*/ |
| **register\_framebuffer**(**struct fb\_info** \*fb\_info)  |--**do\_register\_framebuffer**(**struct fb\_info** \*fb\_info)  1. 获取可用的fb序号: /dev/fb[i]  |--**num\_registered\_fb**++;  |--for (i = 0 ; i < FB\_MAX; i++)  |--if (!**registered\_fb[i])** break;  2. 创建/sys/class/graphics/fb[i]设备及其属性  |--fb\_info->**node = i;**  #define FB\_MAJOR 29 /\* /dev/fb\* framebuffers \*/  |--fb\_info->dev = **device\_create**(fb\_class/\*graphics\*/, fb\_info->device, MKDEV(FB\_MAJOR, i), NULL, "fb%d", i);  |--fb\_init\_device(fb\_info);  |--for (i = 0; i < ARRAY\_SIZE(device\_attrs); i++)  |--device\_create\_file(fb\_info->dev, &device\_attrs[i]);    #define FBPIXMAPSIZE (1024 \* 8)  |--fb\_info->pixmap.addr = kmalloc(FBPIXMAPSIZE, GFP\_KERNEL);  |--fb\_info->pixmap.size = FBPIXMAPSIZE;  |--fb\_info->pixmap.offset = 0;  |--fb\_var\_to\_videomode(&mode, &fb\_info->var);  |--fb\_add\_videomode(&mode, &fb\_info->modelist);  |--registered\_fb[i] = fb\_info;  3. 发出 FB\_EVENT\_FB\_REGISTERED 消息  |--struct fb\_event event;  |--event.info = fb\_info;  |--fb\_notifier\_call\_chain(**FB\_EVENT\_FB\_REGISTERED**, &event);  |--blocking\_notifier\_call\_chain(&fb\_notifier\_list, val, v);  |--notifier\_call\_chain(&nh->head, val, v, nr\_to\_call, nr\_calls); |

## fb\_open()

打开指定的fb\_info设备

|  |
| --- |
| fb\_open(struct inode \*inode, struct file \*file)  |--int fbidx = iminor(inode);  |--struct fb\_info \*info = get\_fb\_info(fbidx);  |--file->private\_data = info;  |--info->fbops->fb\_open(info,1); //owlfb\_open() : do nothing |

## fb\_write() / fb\_read()

|  |
| --- |
| fb\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*ppos)  |--struct fb\_info \*info = file\_fb\_info(file);  1. 分配一页大小的空间地址  |--u8 \* buffer = kmalloc((count > PAGE\_SIZE) ? PAGE\_SIZE : count, GFP\_KERNEL);    //count: buffer的数据大小; cnt: 拷贝的实际数据大小  |--unsigned long p = \*ppos;  |--dst = (u8 \_\_iomem \*) (info->screen\_base + p);  |--int cnt = 0;  |--while (count)  2. 将user空间的数据拷贝到kernel空间  |--c = (count > PAGE\_SIZE) ? PAGE\_SIZE : count;  |--src = buffer;  |--copy\_from\_user(src, buf, c);  3. 将kernel空间的数据拷贝到显存地址中  |--fb\_memcpy\_tofb(dst, src, c);  修正源(buf, count), 目的(dst, cnt)的值  |--dst += c;  |--buf += c;  |--cnt += c;  |--count -= c;  修正显存的剩余空间地址偏移量  |--\*ppos += c;    |--kfree(buffer);  |--return (cnt) ? cnt : err; |
| fb\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*ppos)  |--struct fb\_info \*info = file\_fb\_info(file);  |--buffer = kmalloc((count > PAGE\_SIZE) ? PAGE\_SIZE : count, GFP\_KERNEL);    |--unsigned long p = \*ppos;  |--src = (u8 \_\_iomem \*) (info->screen\_base + p);  |--while (count)  |--c = (count > PAGE\_SIZE) ? PAGE\_SIZE : count;  |--dst = buffer;  |--fb\_memcpy\_fromfb(dst, src, c);  |--dst += c;  |--src += c;    |--copy\_to\_user(buf, buffer, c)  |--\*ppos += c;  |--buf += c;  |--cnt += c;  |--count -= c;  |--kfree(buffer);  |--return (err) ? err : cnt; |

## fb\_ioctl()

|  |
| --- |
| fb\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)  |--do\_fb\_ioctl(info, cmd, arg);  |--switch (cmd)  |--case FBIOGET\_VSCREENINFO: //GET Var  |--struct fb\_var\_screeninfo var = info->var;  |--case FBIOPUT\_VSCREENINFO: //PUT Var  |--case FBIOGET\_FSCREENINFO: //Get fix  |--struct fb\_fix\_screeninfo fix = info->fix;  |--case FBIOPUTCMAP:  |--case FBIOGETCMAP:  |--case FBIOPAN\_DISPLAY:  |--case FBIO\_CURSOR:  |--case FBIOGET\_CON2FBMAP:  |--case FBIOPUT\_CON2FBMAP:  |--case FBIOBLANK:  |--default:  |--info->fbops->fb\_ioctl(info, cmd, arg); //owlfb\_ioctl() : do nothing |

# 1.2 Owlfb设备框架的构建

drivers\video\actions\fb\owlfb.c

module\_init(owlfb\_init);

## owlfb\_init() : 创建注册platform\_driver： owlfb

|  |  |
| --- | --- |
| **owlfb\_init**(void)  1. 注册owlfb设备  |--platform\_driver\_register(&owlfb\_driver);   |  | | --- | | static struct platform\_driver owlfb\_driver = {  .probe = owlfb\_probe,  .remove = owlfb\_remove,  .driver = {  .name = ASOC\_FBDRV\_NAME, //"owlfb"  .owner = THIS\_MODULE,  },  }; | |

## owlfb\_probe(): overlay会创建fb设备

|  |
| --- |
| 调用关系:  owlfb\_probe() : name = "owlfb"  owlfb驱动的创建和注册 :  module\_init(owlfb\_init)  platform\_driver\_register(&owlfb\_driver)  owlfb设备的创建和注册  layers\_default\_init()  ovl = owl\_dss\_get\_overlay(0); // overlay\_list  owl\_overlay\_master\_dev\_set (ovl, "fb", &fb\_pdata)  ovl = owl\_dss\_get\_overlay(3);  owl\_overlay\_master\_dev\_set (ovl, "fb", &fb\_pdata\_extra)  ovl->master\_pdev = platform\_device\_alloc(“owlfb”, ovl->id);  platform\_device\_add\_data(ovl->master\_pdev, pdata, pdata\_size); |

|  |  |
| --- | --- |
| **owlfb\_probe**(struct platform\_device \*pdev)  1. 创建owlfb\_device结构体，包含fb\_info结构体  |--**struct owlfb\_device** \***fbdev** = **kzalloc**(sizeof(struct owlfb\_device), GFP\_KERNEL);  2. 创建并初始化fb\_info结构体  |--struct owlfb\_pdata \*pdata = pdev->dev.platform\_data;  |--**owlfb**\_**create**\_**framebuffer**(fbdev, pdata->overlay\_id, pdata);  2.1分配size = fb\_info+owlfb\_info的空间  |--**struct** **fb**\_**info** \***fbi** = **framebuffer**\_**alloc**(sizeof(struct owlfb\_info), fbdev->dev);  |--clear\_fb\_info(fbi);  |--strlcpy(fbi->fix.id, **“owlfd”**, sizeof(fbi->fix.id));  |--ofbi = FB2OFB(fbi);  |--ofbi->overlays[0] = owl\_dss\_get\_overlay(ovl\_id);  |--**fbdev->fbi = fbi;**  2.2 分配fb的screen\_base地址  |--owlfb\_allocate\_fb\_mem(fbdev, use\_rsvmem);  |--owlfb\_get\_rsvmem(fbi, size);  |--rg = &ofbi->region;  |--fbi->**screen**\_**base** = rg->vaddr = vmap(rg->pages, nr\_pages, VM\_MAP, pgprot);  |--fbi->fix.smem\_len = rg->size = map\_size;  |--fbi->fix.smem\_start = rg->paddr = owlfb\_rsvmem\_start  2.3 设置fb的ops操作函数  |--owlfb\_fb\_init(fbdev, fbdev->fbi, pdata->inited);  |--fbi->fbops = &**owlfb**\_**ops**;   |  | | --- | | static **struct fb\_ops** **owlfb**\_**ops** = {  .owner = THIS\_MODULE,  .fb\_open = owlfb\_open,  .fb\_release = owlfb\_release,  .fb\_blank = owlfb\_blank,  .fb\_ioctl = owlfb\_ioctl,  .fb\_check\_var = owlfb\_check\_var,  .fb\_set\_par = owlfb\_set\_par,  .fb\_pan\_display = owlfb\_pan\_display,  .fb\_setcolreg = owlfb\_setcolreg,  }; |   2.4 注册fb\_info结构体到fb\_class子系统和中  |--**register\_framebuffer**(fbdev->fbi);    2.5 创建/sys/devices/platform/owlfb.0/graphics/fb0设备的连接  |--owlfb\_create\_overlay\_syslink(fbdev->fbi);  |--owlfb\_init\_attr(fbdev->fbi);    3. 设置owlfb的回调函数，处理以下事件:  //ASOC\_DSS\_EVENT\_OVL\_DISPLAY\_SET, ASOC\_DSS\_EVENT\_OVL\_DISPLAY\_UPDATE,  //ASOC\_DSS\_EVENT\_OVL\_CHANNEL\_SET, ASOC\_DSS\_EVENT\_OVL\_ENABLE\_REQ  |--**owlfb**\_**register**\_**dss**(fbdev);  |--fbdev->owl\_dss\_notif.notifier\_call = **owl**\_**dss**\_**notifier**\_**callback**;  |--owl\_dss\_register\_client(&fbdev->owl\_dss\_notif);    4. 显示开机画面  |--fbi = fbdev->fbi;  |--memset(fbi->screen\_base, 0, fbi->var.yres \* fbi->fix.line\_length);  |--fb\_prepare\_logo(fbi, FB\_ROTATE\_UR);  |--**fb**\_**show**\_**logo**(fbi, FB\_ROTATE\_UR); |

# 2.1 Owl\_display

drivers\video\actions\display\owl\_display.c

**postcore\_initcall**(owl\_display\_class\_init);

|  |
| --- |
| Owl\_display子系统下挂载了3个显示设备  ==============================  /sys/class/**owl**\_**display**/  ==============================  lrwxrwxrwx root root **HDMI** -> ../../devices/virtual/owl\_display/HDMI  lrwxrwxrwx root root **cvbs** -> ../../devices/virtual/owl\_display/cvbs  lrwxrwxrwx root root **owl\_dummy\_displayer** -> ../../devices/virtual/owl\_display/owl\_dummy\_displayer |

## owl\_display\_class\_init(): 创建owl\_display子系统

|  |  |
| --- | --- |
| **owl**\_**display**\_**class**\_**init**(void)  1. 创建注册class子系统 : /sys/class/owl\_display  |--**class\_register**(&**owl\_display\_class**);   |  | | --- | | static **struct** **class** **owl**\_**display**\_**class** = {  .name = "owl\_display",  .dev\_release = owl\_display\_release,  .dev\_attrs = owl\_display\_attributes,  }; |   2. 注册dummy显示设备: owl\_dummy\_displayer  |--**owl**\_**display**\_**dummy**\_**device**\_**register**(); |

## owl\_display\_device\_register() : 创建注册owl\_display\_device显示设备

|  |
| --- |
| **LIST**\_**HEAD**(**display\_list**);  所有的owl\_display\_device结构体设备都挂载到这个链表中 |
| **owl**\_**display**\_**device**\_**register**(const char \***name**,  struct device \*dev,  void \*devdata,  int **display**\_**id**,  const struct owl\_videomode \***modes**,  u32 num\_modes,  struct display\_ops \***ops**)  {  1. 创建owl\_display\_device设备并初始化  |--**struct owl\_display\_device \*new\_dev;**  |--new\_dev = **kzalloc**(sizeof(struct owl\_display\_device), GFP\_KERNEL);  |--new\_dev->display\_id = **display**\_**id**;  |--new\_dev->**ops** = **ops**;  |--owl\_videomode\_to\_modelist(modes, num\_modes, &new\_dev->modelist);  |--new\_dev->**modes** = **modes**;  |--new\_dev->num\_modes = num\_modes;  |--new\_dev->**disp**\_**cur**\_**mode** = &modelist->mode;  |--new\_dev->**fb**\_**cur**\_**mode** = new\_dev->disp\_cur\_mode;  2. 注册owl\_display\_device到 display\_list链表中  |--**owl**\_**display**\_**list**\_**register**(new\_dev);  |--list\_add\_tail(&disp\_dev->entry, &**display\_list**);  3. 注册owl\_display\_device到/sys/devices/virtual/owl\_display中  |--new\_dev->dev.class = &**owl**\_**display**\_**class**;  |--dev\_set\_name(&new\_dev->dev, name);  |--**device\_register**(&new\_dev->dev); |

## owl\_display\_connect(display\_id) : 获取显示设备

|  |
| --- |
| owl\_display\_connect(HDMI\_DISPLAYER);  owl\_display\_connect(TV\_CVBS\_DISPLAYER); |
| struct owl\_display\_device \***owl\_display\_connect**(int display\_id)  |--struct owl\_display\_device \*disp\_dev = get\_owl\_display(display\_id);  |-- list\_for\_each\_entry(disp\_dev, &display\_list, entry)  |-- if (disp\_dev->display\_id == display\_id) return disp\_dev;  |--disp\_dev->ops->update\_status(disp\_dev);  |--return disp\_dev; |

# owl\_display\_dummy\_device

## owl\_display\_dummy\_device\_register(): 创建dummy显示设备

|  |
| --- |
| **owl**\_**display**\_**dummy**\_**device**\_**register**();  |--**struct owl\_display\_device** \*disp\_dev;  |--disp\_dev = **owl**\_**display**\_**device**\_**register**("**owl**\_**dummy**\_**displayer**", NULL,  NULL, **DUMMY**\_**DISPLAYER**, dummy\_displayer\_**modes**,  ARRAY\_SIZE(dummy\_displayer\_modes), &**dummy**\_**device**\_**ops**);  |--disp\_dev->disp\_cur\_mode = &dummy\_displayer\_modes[0].mode;  |--disp\_dev->is\_plugged = 1; |

## owl\_videomode

|  |
| --- |
| static const **struct** **owl**\_**videomode** **dummy**\_**displayer**\_**modes**[] = {  [0] = {  .valid = 0,  .mode = {  .name = NULL,  .refresh = 50,  .xres = 1024,  .yres = 576,  .pixclock = 0, /\*in pico second, 0.000 000 000 001s\*/  .left\_margin = 0,  .right\_margin = 0,  .upper\_margin = 0,  .lower\_margin = 0,  .hsync\_len = 0,  .vsync\_len = 0,  .sync = 0,  .vmode = FB\_VMODE\_NONINTERLACED,  .flag = FB\_MODE\_IS\_STANDARD,  .vid = 1,  },  },  }; |

## display\_ops

|  |
| --- |
| static **struct** **display**\_**ops** **dummy**\_**device**\_**ops** = {  .get\_status = dummy\_device\_get\_status,  .get\_devclk\_khz = dummy\_device\_get\_devclk\_khz,  }; |

# End