

FrameBuffer驱动程序分析

原创

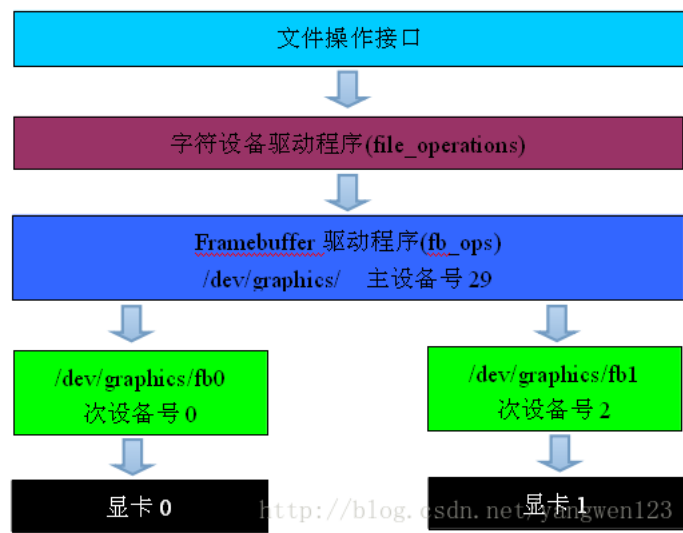
2013年09月28日 21:17:32

标签

Android / Linux / FrameBuffer / fb_info

6202

FrameBuffer通常作为LCD控制器或者其他显示设备的驱动，FrameBuffer驱动是一个字符设备，设备节点是/dev/fbX，主设备号为29，次设备号递增。可以将Framebuffer看成是显示内存的一个映像，将其映射到进程地址空间之后，就可以直接进行读写操作。而写操作可以立即反应在屏幕上。这种操作是抽象的，统一的。用户不必关心物理显存的位置、换页机制等具体细节。这些都是由Framebuffer设备驱动来完成的。Framebuffer设备为上层应用程序提供系统调用，也为下一层的特定硬件驱动提供接口；那些底层硬件驱动需要用到这儿的接口来向系统内核注册它们自己。



Linux中的PCI设备可以将其控制寄存器映射到物理内存空间，而后，对这些控制寄存器的访问变成了对内存的访问，因此，这些寄存器又被称为"memio"。一旦被映射到物理内存，Linux的普通进程就可以通过mmap将这些内存I/O映射到进程地址空间，这样就可以直接访问这些寄存器了。

FrameBuffer设备属于字符设备，采用了一驱多层的接口方式，Linux为帧缓冲设备定义了驱动层的接口fb_info结构，在文件层上，用户调用file_operations的函数操作，间接调用fb_info中的fb_ops函数集来操作硬件。

Framebuffer数据结构

kernel\include\linux\fb.h

fb_info是Linux为帧缓冲设备定义的驱动层接口。它不仅包含了底层函数，而且还有记录设备状态的数据。每个帧缓冲设备都与一个fb_info结构相对应。

```
[cpp]
1. struct fb_info {
2.     atomic_t count;
3.     int node; /* 一个FrameBuffer设备的次设备号*/
4.     int flags;
5.     struct mutex lock; /* Lock for open/release/ioctl funcs */
6.     struct mutex mm_lock; /* Lock for fb_mmap and smem_* fields */
7.     struct fb_var_screeninfo var; /* Current var */
8.     struct fb_fix_screeninfo fix; /* Current fix */
9.     struct fb_monspecs monspecs; /* Current Monitor specs */
10.    struct work_struct queue; /* Framebuffer event queue */
11.    struct fb_pixmap pixmap; /* Image hardware mapper */
12.    struct fb_pixmap sprite; /* Cursor hardware mapper */
13.    struct fb_cmap cmap; /* Current cmap */
}
```



快乐安卓

原创	粉丝	喜欢	评论
159	763	196	16

等级： 博客 5 访问量： 71万+

积分： 8494 排名： 2907

广告

博主最新文章

- AndroidO Treble架构下HIDL服务Java实现
- AndroidO Treble架构下HIDL服务查询过程
- AndroidO Treble架构下Transport类型查询过程
- AndroidO Treble架构下hwserVICemanager启动过程
- AndroidO Treble架构下Binder对象的换过程

文章分类

- 【Android 系统分析】
- 【Android 应用开发】
- 【Ubuntu系统配置】
- 【Java 开发】
- 【Makefile 学习】
- 【git版本管理】

展开

文章存档

- 2018年4月
- 2015年1月
- 2014年12月
- 2014年11月
- 2014年10月
- 2014年9月

展开

博主热门文章

Android之rild进程启动源码分析

```
17. struct backlight_device *bl_dev;
18. /* Backlight level curve */
19. struct mutex bl_curve_mutex;
20. u8 bl_curve[FB_BACKLIGHT_LEVELS];
21. #endif
22. #ifdef CONFIG_FB_DEFERRED_IO
23. struct delayed_work deferred_work;
24. struct fb_deferred_io *fbdefio;
25. #endif
26. struct fb_ops *fbops;
27. struct device *device; /* This is the parent */
28. struct device *dev; /* This is this fb device */
29. int class_flag; /* primary flags */
30. #ifdef CONFIG_FB_TILEBLITTING
31. struct fb_tile_ops *tileops; /* tile Blitting */
32. #endif
33. char __iomem *screen_base; /* physical address */
34. unsigned long screen_size; /* Amount of ioremapped VRAM or 0 */
35. void *pseudo_palette; /* palette of 16 colors */
36. #define FBINFO_STATE_RUNNING 0
37. #define FBINFO_STATE_SUSPENDED 1
38. u32 state; /* Hardware state i.e suspend */
39. void *fbcon_par; /* fbcon use-only private area */
40. void *par;
41. struct apertures_struct {
42. unsigned int count;
43. struct aperture {
44. resource_size_t base;
45. resource_size_t size;
46. } ranges[0];
47. } *apertures;
48. };
```

fb_var_screeninfo：用于记录用户可修改的显示控制器参数，包括屏幕分辨率、每个像素点的比特数等

```
[cpp]
1. struct fb_var_screeninfo {
2.     __u32 xres; /* 行可见像素*/
3.     __u32 yres; /* 列可见像素*/
4.     __u32 xres_virtual; /* 行虚拟像素*/
5.     __u32 yres_virtual; /* 列虚拟像素*/
6.     __u32 xoffset; /* 水平偏移量*/
7.     __u32 yoffset; /* 垂直偏移量*/
8.     __u32 bits_per_pixel; /*每个像素所占bit位数*/
9.     __u32 grayscale; /* 灰色刻度*/
10. struct fb_bitfield red; /* bitfield in fb mem if true color, */
11. struct fb_bitfield green; /* else only length is significant */
12. struct fb_bitfield blue;
13. struct fb_bitfield transp; /* transparency */
14. __u32 nonstd; /* != 0 Non standard pixel format */
15. __u32 activate; /* see FB_ACTIVATE_* */
16. __u32 height; /* 图像高度*/
17. __u32 width; /* 图像宽度*/
18. __u32 accel_flags; /* (OBSOLETE) see fb_info.flags */
19. __u32 pixclock; /* pixel clock in ps (pico seconds) */
20. __u32 left_margin; /* time from sync to picture */
21. __u32 right_margin; /* time from picture to sync */
22. __u32 upper_margin; /* time from sync to picture */
23. __u32 lower_margin;
24. __u32 hsync_len; /* length of horizontal sync */
25. __u32 vsync_len; /* length of vertical sync */
26. __u32 sync; /* see FB_SYNC_* */
27. __u32 vmode; /* see FB_VMODE_* */
28. __u32 rotate; /* angle we rotate counter clockwise */
29. __u32 reserved[5]; /* Reserved for future compatibility */
30. };
```

fb_fix_screeninfo：记录了用户不能修改的显示控制器的参数，这些参数是在驱动初始化时设置的

```
[cpp]
1. struct fb_fix_screeninfo {
2.     char id[16]; /* identification string eg "TT Builtin" */
3.     unsigned long smem_start; /* Start of frame buffer mem */
4.     __u32 smem_len; /* Length of frame buffer mem */
5.     __u32 type; /* see FB_TYPE_* */
6.     __u32 type_aux; /* Interleave for interleaved Planes */
7. };
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

Android Service之MountService源码分析

21720

Android 匿名共享内存C接口分析

15547

Android系统Choreographer机制实现

15065

Android显示系统设计框架介绍

14770

深入剖析Android音频之AudioTrack

14146

深入剖析Android音频之AudioPolicyManager

12031

Android Init进程源码分析

12004

Android系统Audio框架介绍

11970

PendingIntent 用法深入理解

11130

联系我们



请扫描二维码联系客服

webmaster@csdn

400-660-0108

QQ客服 客服论

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

```

10.     __u16 ywrapstep;          /* zero if no hardware ywrap */
11.     __u32 line_length;        /* length of a line in bytes */
12.     unsigned long mmio_start; /* Start of Memory Mapped I/O */
13.     __u32 mmio_len;           /* Length of Memory Mapped I/O */
14.     __u32 accel;              /* Indicate to driver which */
15.     __u16 reserved[3];        /* Reserved for future compatibility */
16. };

```

fb_ops是提供给底层设备驱动的一个接口编程的套路，填写fb_ops结构体。



我们编写一个Framebuffer的时候，就要依照Linux FrameBuffer

[cpp]

```

1. struct fb_ops {
2.     /* open/release and usage mark */
3.     struct module *owner;
4.     int (*fb_open)(struct fb_info *info, int user);
5.     int (*fb_release)(struct fb_info *info, int user);
6.
7.     /* For framebuffers with strange non linear layouts or that do not
8.      * work with normal memory mapped access
9.      */
10.    ssize_t (*fb_read)(struct fb_info *info, char __user *buf,
11.        size_t count, loff_t *ppos);
12.    ssize_t (*fb_write)(struct fb_info *info, const char __user *buf,
13.        size_t count, loff_t *ppos);
14.
15.    /* checks var and eventually tweaks it to something supported,
16.     * DO NOT MODIFY PAR */
17.    int (*fb_check_var)(struct fb_var_screeninfo *var, struct fb_info *info);
18.
19.    /* set the video mode according to info->var */
20.    int (*fb_set_par)(struct fb_info *info);
21.
22.    /* set color register */
23.    int (*fb_setcolreg)(unsigned regno, unsigned red, unsigned green,
24.        unsigned blue, unsigned transp, struct fb_info *info);
25.
26.    /* set color registers in batch */
27.    int (*fb_setcmap)(struct fb_cmap *cmap, struct fb_info *info);
28.
29.    /* blank display */
30.    int (*fb_blank)(int blank, struct fb_info *info);
31.
32.    /* pan display */
33.    int (*fb_pan_display)(struct fb_var_screeninfo *var, struct fb_info *info);
34.
35.    /* Draws a rectangle */
36.    void (*fb_fillrect) (struct fb_info *info, const struct fb_fillrect *rect);
37.    /* Copy data from area to another */
38.    void (*fb_copyarea) (struct fb_info *info, const struct fb_copyarea *region);
39.    /* Draws a image to the display */
40.    void (*fb_imageblit) (struct fb_info *info, const struct fb_image *image);
41.
42.    /* Draws cursor */
43.    int (*fb_cursor) (struct fb_info *info, struct fb_cursor *cursor);
44.
45.    /* Rotates the display */
46.    void (*fb_rotate)(struct fb_info *info, int angle);
47.
48.    /* wait for blit idle, optional */
49.    int (*fb_sync)(struct fb_info *info);
50.
51.    /* perform fb specific ioctl (optional) */
52.    int (*fb_ioctl)(struct fb_info *info, unsigned int cmd,
53.        unsigned long arg);
54.
55.    /* Handle 32bit compat ioctl (optional) */
56.    int (*fb_compat_ioctl)(struct fb_info *info, unsigned cmd,
57.        unsigned long arg);
58.
59.    /* perform fb specific mmap */
60.    int (*fb_mmap)(struct fb_info *info, struct vm_area_struct *vma);
61.
62.    /* get capability given var */
63.    void (*fb_get_caps)(struct fb_info *info, struct fb_blit_caps *caps,
64.        struct fb_var_screeninfo *var);
65. };

```



```
4.     .read      = seq_read,
5.     .llseek    = seq_lseek,
6.     .release    = seq_release,
7. };
```

因此可以对/proc/fb文件进行打开，读写操作。然后注册一个主设备号为29的字符设备，fbmem_init函数中注册了字符设备的文件操作接口函数fb_fops，定义如下：

```
[cpp]
1. static const struct file_operations fb_fops = {
2.     .owner = THIS_MODULE,
3.     .read = fb_read,
4.     .write = fb_write,
5.     .unlocked_ioctl = fb_ioctl,
6. #ifdef CONFIG_COMPAT
7.     .compat_ioctl = fb_compat_ioctl,
8. #endif
9.     .mmap = fb_mmap,
10.    .open = fb_open,
11.    .release = fb_release,
12. #ifdef HAVE_ARCH_FB_UNMAPPED_AREA
13.    .get_unmappable_area = get_fb_unmappable_area,
14. #endif
15. #ifdef CONFIG_FB_DEFERRED_IO
16.    .fsync = fb_deferred_io_fsync,
17. #endif
18.    .llseek = default_llseek,
19. };
```

Framebuffer驱动注册过程

变量定义：

```
[cpp]
1. //保存注册的所有Framebuffer驱动
2. extern struct fb_info *registered_fb[FB_MAX];
3. //已注册的Framebuffer驱动的个数
4. extern int num_registered_fb;
```

任何一个特定硬件Framebuffer驱动在初始化时都必须向fbmem.c注册，Framebuffer模块提供了驱动注册接口函数register_framebuffer：

```
[cpp]
1. int register_framebuffer(struct fb_info *fb_info)
2. {
3.     int ret;
4.     mutex_lock(&istration_lock);
5.     ret = do_register_framebuffer(fb_info);
6.     mutex_unlock(&istration_lock);
7.     return ret;
8. }
```

参数fb_info描述特定硬件的Framebuffer驱动信息

```
[cpp]
1. static int do_register_framebuffer(struct fb_info *fb_info)
2. {
3.     int i;
4.     struct fb_event event;
5.     struct fb_videomode mode;
6.     if (fb_check_foreignness(fb_info))
7.         return -ENOSYS;
8.     //根据当前注册的fb_info的apertures属性从Framebuffer驱动数组registered_fb中查询是否存在冲突
9.     do_remove_conflicting_framebuffers(fb_info->apertures, fb_info->fix.id,
10.         fb_is_primary_device(fb_info));
11.     //判断已注册的驱动是否超过32个Framebuffer驱动
12.     if (num_registered_fb == FB_MAX)
13.         return -ENXIO;
14.     //增加已注册的驱动个数
15.     num_registered_fb++;
16.     //从数组registered_fb中查找空闲元素，用于存储当前注册的fb_info
17.     for (i = 0 ; i < FB_MAX; i++)
```

```

21.     fb_info->node = i;
22.     //初始化当前注册的fb_info的成员信息
23.     atomic_set(&fb_info->count, 1);
24.     mutex_init(&fb_info->lock);
25.     mutex_init(&fb_info->mm_lock);
26.     //在/dev目录下创建一个fbx的设备文件，次设备号就是该fb_info在数组registered_fb中的索引
27.     fb_info->dev = device_create(fb_class, fb_info->device, MKDEV(FB_MAJOR, i), NULL, "fb%d", i);
28.     if (IS_ERR(fb_info->dev)) {
29.         printk(KERN_WARNING "Unable to create device for framebuffer %d; errno = %ld\n", i, PTR_ERR(fb_info
->dev));
30.         fb_info->dev = NULL;
31.     } else
32.         //初始化fb_info
33.         fb_init_device(fb_info);
34.     if (fb_info->pixmap.addr == NULL) {
35.         fb_info->pixmap.addr = kmap(BPIXMAPSIZE, GFP_KERNEL);
36.         if (fb_info->pixmap.addr)
37.             fb_info->pixmap.size = FBPIXMAPSIZE;
38.             fb_info->pixmap.buf_al: 1;
39.             fb_info->pixmap.scan_a: 1;
40.             fb_info->pixmap.access_align = 32;
41.             fb_info->pixmap.flags = FB_PIXMAP_DEFAULT;
42.         }
43.     }
44.     fb_info->pixmap.offset = 0;
45.     if (!fb_info->pixmap.blit_x)
46.         fb_info->pixmap.blit_x = ~(u32)0;
47.     if (!fb_info->pixmap.blit_y)
48.         fb_info->pixmap.blit_y = ~(u32)0;
49.     if (!fb_info->modelist.prev || !fb_info->modelist.next)
50.         INIT_LIST_HEAD(&fb_info->modelist);
51.     fb_var_to_videomode(&mode, &fb_info->var);
52.     fb_add_videomode(&mode, &fb_info->modelist);
53.     //将特定硬件对应的fb_info注册到registered_fb数组中
54.     registered_fb[i] = fb_info;
55.     event_info = fb_info;
56.     if (!lock_fb_info(fb_info))
57.         return -ENODEV;
58.     //使用Linux事件通知机制发送一个Framebuffer注册事件FB_EVENT_FB_REGISTERED
59.     fb_notifier_call_chain(FB_EVENT_FB_REGISTERED, &event);
60.     unlock_fb_info(fb_info);
61.     return 0;
62. }

```

注册过程就是将指定的设备驱动信息fb_info存放到registered_fb数组中。因此在注册具体的fb_info时，首先要构造一个fb_info数据结构，并初始化该数据结构，该结构用于描述一个特定的Framebuffer驱动。

fbX设备文件的打开过程

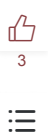
open("/dev/fb0")打开设备文件fb0对应的操作过程如下：

```

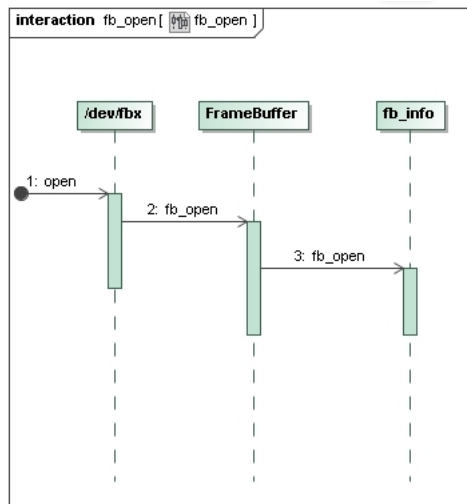
[cpp]
1. static int fb_open(struct inode *inode, struct file *file)
2. __acquires(&info->lock)
3. __releases(&info->lock)
4. {
5.     //从文件节点中取得次设备号
6.     int fbidx = iminor(inode);
7.     struct fb_info *info;
8.     int res = 0;
9.     //根据次设备号从registered_fb数组中取出对应的fb_info
10.    info = get_fb_info(fbidx);
11.    if (!info) {
12.        request_module("fb%d", fbidx);
13.        info = get_fb_info(fbidx);
14.        if (!info)
15.            return -ENODEV;
16.    }
17.    if (IS_ERR(info))
18.        return PTR_ERR(info);
19.    mutex_lock(&info->lock);
20.    if (!try_module_get(info->fbops->owner)) {
21.        res = -ENODEV;
22.        goto out;
23.    }
24.    //将当前的fb_info保存到/dev/fbx设备文件的private_data成员中

```

```
28.         res = info->fbops->fb_open(info,1);
29.         if (res)
30.             module_put(info->fbops->owner);
31.     }
32. #ifdef CONFIG_FB_DEFERRED_IO
33.     if (info->fbdefio)
34.         fb_deferred_io_open(info, inode, file);
35. #endif
36. out:
37.     mutex_unlock(&info->lock);
38.     if (res)
39.         put_fb_info(info);
40.     return res;
41. }
```



打开过程很简单，首先从文件节点中取出FrameBuffer的fb_info数据信息，并保存到设备文件file的private_data变量中，然后调用当前fb_info的fb_open函数完成设备打开过程。该函数在构造具体的fb_info并注册Framebuffer时，就已注册了对应的打开函数指针。



<http://blog.csdn.net/yangwen123>

fbX设备文件的映射过程

```
[cpp]
1. static int fb_mmap(struct file *file, struct vm_area_struct * vma)
2. {
3.     //从文件节点中取出fb_info, 并且判断是否和private_data变量中的fb_info相同
4.     struct fb_info *info = file_fb_info(file);
5.     struct fb_ops *fb;
6.     unsigned long off;
7.     unsigned long start;
8.     u32 len;
9.     if (!info)
10.         return -ENODEV;
11.     if (vma->vm_pgoff > (~0UL >> PAGE_SHIFT))
12.         return -EINVAL;
13.     off = vma->vm_pgoff << PAGE_SHIFT;
14.     fb = info->fbops;
15.     if (!fb)
16.         return -ENODEV;
17.     mutex_lock(&info->mm_lock);
18.     //如果fb_info中注册了fb_mmap函数, 则调用fb_info中的fb_mmap来完成地址空间映射
19.     if (fb->fb_mmap) {
20.         int res;
21.         res = fb->fb_mmap(info, vma);
22.         mutex_unlock(&info->mm_lock);
23.         return res;
24.     }
25.     //如果具体的fb_info没有实现fb_mmap
26.     start = info->fix.smem_start;
27.     len = PAGE_ALIGN((start & ~PAGE_MASK) + info->fix.smem_len);
28.     if (off >= len) {
29.         /* memory mapped io */

```



```
33.         return -EINVAL;
34.     }
35.     start = info->fix.mmio_start;
36.     len = PAGE_ALIGN((start & ~PAGE_MASK) + info->fix.mmio_len);
37. }
38. mutex_unlock(&info->mm_lock);
39. start &= PAGE_MASK;
40. if ((vma->vm_end - vma->vm_start + off) > len)
41.     return -EINVAL;
42. off += start;
43. vma->vm_pgoff = off >> PAGE_SHIFT;
44. /* This is an IO map - tell maydump to skip this VMA */
45. vma->vm_flags |= VM_IO | VM_RESERVED;
46. vma->vm_page_prot = vm_get_page_prot(vma->vm_flags);
47. fb_pgprotect(file, vma, off);
48. if (io_remap_pfn_range(vma, vm_start, vma->vm_pgoff,
49.                        vma->vm_end - vma->vm_start, vma->vm_page_prot))
50.     return -EAGAIN;
51. return 0;
52. }
```

这里和fb打开过程类似，仍然是调用具体的fb_info的映射函数来完成地址空间映射过程，但是也有区别，就是在具体的fb_info没有实现地址空间映射时，就在Framebuffer这一层完成映射过程。

fbX设备文件的命令控制过程

Framebuffer	Fb_info
FBIOGET_VSCREENINFO	
FBIOPUT_VSCREENINFO	
FBIOGET_FSCREENINFO	
FBIOPUTCMAP	
FBIOGETCMAP	
FBIOPAN_DISPLAY	
FBIO_CURSOR	
FBIOGET_CON2FBMAP	
FBIOPUT_CON2FBMAP	
FBIOBLANK	
	fb->fb_io

```
[cpp]
1. static long do_fb_ioctl(struct fb_info *info, unsigned int cmd,unsigned long arg)
2. {
3.     struct fb_ops *fb;
4.     struct fb_var_screeninfo var;
5.     struct fb_fix_screeninfo fix;
6.     struct fb_con2fbmap con2fb;
7.     struct fb_cmap cmap_from;
8.     struct fb_cmap_user cmap;
9.     struct fb_event event;
10.    void __user *argp = (void __user *)arg;
11.    long ret = 0;
12.    switch (cmd) {
13.    case FBIOGET_VSCREENINFO:
14.        if (!lock_fb_info(info))
15.            return -ENODEV;
16.        var = info->var;
17.        unlock_fb_info(info);
18.
19.        ret = copy_to_user(argp, &var, sizeof(var)) ? -EFAULT : 0;
20.        break;
21.    case FBIOPUT_VSCREENINFO:
22.        if (copy_from_user(&var, argp, sizeof(var)))
23.            return -EFAULT;
24.        if (!lock_fb_info(info))
25.            return -ENODEV;
26.        console_lock();
27.        info->flags |= FBINFO_MISC_USEREVENT;
```



```

31.         unlock_fb_info(info);
32.         if (!ret && copy_to_user(argp, &var, sizeof(var)))
33.             ret = -EFAULT;
34.         break;
35.     case FBIOGET_FSCREENINFO:
36.         if (!lock_fb_info(info))
37.             return -ENODEV;
38.         fix = info->fix;
39.         unlock_fb_info(info);
40.
41.         ret = copy_to_user(argp, &fix, sizeof(fix)) ? -EFAULT : 0;
42.         break;
43.     case FBIOPUTCMAP:
44.         if (copy_from_user(&cmap, &user_cmap, sizeof(cmap)))
45.             return -EFAULT;
46.         ret = fb_set_user_cmap(&cmap, info);
47.         break;
48.     case FBIOGETCMAP:
49.         if (copy_from_user(&cmap, &user_cmap, sizeof(cmap)))
50.             return -EFAULT;
51.         if (!lock_fb_info(info))
52.             return -ENODEV;
53.         cmap_from = info->cmap;
54.         unlock_fb_info(info);
55.         ret = fb_cmap_to_user(&cmap_from, &cmap);
56.         break;
57.     case FBIOPAN_DISPLAY:
58.         if (copy_from_user(&var, argp, sizeof(var)))
59.             return -EFAULT;
60.         if (!lock_fb_info(info))
61.             return -ENODEV;
62.         console_lock();
63.         ret = fb_pan_display(info, &var);
64.         console_unlock();
65.         unlock_fb_info(info);
66.         if (ret == 0 && copy_to_user(argp, &var, sizeof(var)))
67.             return -EFAULT;
68.         break;
69.     case FBIO_CURSOR:
70.         ret = -EINVAL;
71.         break;
72.     case FBIOGET_CON2FBMAP:
73.         if (copy_from_user(&con2fb, argp, sizeof(con2fb)))
74.             return -EFAULT;
75.         if (con2fb.console < 1 || con2fb.console > MAX_NR_CONSOLES)
76.             return -EINVAL;
77.         con2fb.framebuffer = -1;
78.         event.data = &con2fb;
79.         if (!lock_fb_info(info))
80.             return -ENODEV;
81.         event.info = info;
82.         fb_notifier_call_chain(FB_EVENT_GET_CONSOLE_MAP, &event);
83.         unlock_fb_info(info);
84.         ret = copy_to_user(argp, &con2fb, sizeof(con2fb)) ? -EFAULT : 0;
85.         break;
86.     case FBIOPUT_CON2FBMAP:
87.         if (copy_from_user(&con2fb, argp, sizeof(con2fb)))
88.             return -EFAULT;
89.         if (con2fb.console < 1 || con2fb.console > MAX_NR_CONSOLES)
90.             return -EINVAL;
91.         if (con2fb.framebuffer < 0 || con2fb.framebuffer >= FB_MAX)
92.             return -EINVAL;
93.         if (!registered_fb[con2fb.framebuffer])
94.             request_module("fb%d", con2fb.framebuffer);
95.         if (!registered_fb[con2fb.framebuffer]) {
96.             ret = -EINVAL;
97.             break;
98.         }
99.         event.data = &con2fb;
100.         if (!lock_fb_info(info))
101.             return -ENODEV;
102.         event.info = info;
103.         ret = fb_notifier_call_chain(FB_EVENT_SET_CONSOLE_MAP, &event);
104.         unlock_fb_info(info);
105.         break;
106.     case FBIOBLANK:
107.         if (!lock_fb_info(info))
108.             return -ENODEV;

```

```
112.         info->flags &= ~FBINFO_MISC_USEREVENT;
113.         console_unlock();
114.         unlock_fb_info(info);
115.         break;
116.     default:
117.         if (!lock_fb_info(info))
118.             return -ENODEV;
119.         fb = info->fbops;
120.         if (fb->fb_ioctl)
121.             ret = fb->fb_ioctl(info, arg);
122.         else
123.             ret = -ENOTTY;
124.         unlock_fb_info(info);
125.     }
126.     return ret;
127. }
```

Framebuffer驱动的框架就介绍到这里，总结一下：

- 1) 构建一个fb_info数据结构，用来描述设备；
- 2) 调用Framebuffer驱动模块提供的接口函数register_framebuffer来注册帧缓冲设备；
- 3) 对Framebuffer设备文件的操作过程是，首先执行Framebuffer驱动函数，然后根据注册的帧缓冲设备的设备号得到注册的fb_info，最后调用具体的帧缓冲设备的操作函数；

目前您尚未登录，请 [登录](#) 或 [注册](#) 后参与评论

-  kris_fei 2017-11-03 09:54 #2楼 [回复](#)
- 学习了，感谢分享！
-  gzzaigcn 2014-01-07 13:51 #1楼 [回复](#)
- 问下为何android是/dev/graphics/fb0，而纯linux对应的是/dev/fb0.

基于framebuffer的驱动分析 qq_28992301 2016年10月03日 10:47 2213

framebuffer帧缓冲（简称fb）是linux内核中虚拟出的一个设备，是一个platform类型设备，设备文件位于/dev/fb*...

Linux Framebuffer驱动剖析之二—驱动框架、接口实现和使用




















本文继上一篇文章《Linux Framebuffer驱动剖析之一—软件需求》，深入分析LinuxFramebuffer子系统的驱动框架、接口实现和使用。...

yueqian_scut 2015年12月29日 22:16 3871

Linux Framebuffer 驱动框架之一概念介绍及LCD硬件原理

一、基本概念 帧缓冲（Framebuffer）是Linux系统为显示设备提供的一个接口，它将显示缓冲区抽象，屏蔽图像硬件的底层差异，允许上层应用程序在图形模式下直接对显示缓冲区进行读写操作。用...

gqb666 2013年07月18日 23:07 14848

<h3>LCD驱动调试以及Framebuffer</h3> <p>内容提要: 1. android display相关的名词 2. 调试LCD驱动需要注意的步骤 3. 关于帧缓冲区分及I/O内存 ----- -----...</p>		 alifrank	2015年08月27日 14:53	 1764
<h3>linux LCD驱动（二）--Framebuffer</h3> <p>2. Linux 驱动 2.1 FrameBuffer Linux是工作在 模式下，所以用户态进程是无法像DOS那样使用显卡BIOS里提供的中断调用来实现直接写屏，Lin仿显卡的功能，将显u...</p>		 jmq_0000	2011年12月27日 09:23	 19305
<h3>Framebuffer驱动程序分析</h3> <p>Framebuffer通常作为LCD控制器或者其他显示 驱动，Framebuffer驱动是一个字符设备，设备节点是/dev/fbX，主设备号为29，次设备号递增，用户可以将Framebuffer看...</p>		 yangwen123	2013年09月28日 21:17	 6202
<h3>68 linux framebuffer设备驱动之spi lcd屏驱动</h3> <p>前面驱动的spi lcd仅仅是刷了一下图而已， 如果要让QT图形程序在此lcd上显示的话，还需要实现标准的framebuffer设备驱动才可以实现一个fb设备驱动好， QT程序就可以在显存里显示出来...</p>		 jklinux	2017年07月06日 11:41	 1898
<h3>2016/1/9：深度剖析安卓Framebuffer设备驱动</h3> <p>忙了几天，今天在公司居然没什么活干，所以早上就用公司的电脑书写之前在公司编写framebuffer的使用心得体会总结，这也算是一点开发经验，不过我还没写完，精华部分还是自己藏着吧。直到下午才开始有点...</p> <div> morixinguan</div> <div>2016年01月08日 21:26</div> <div> 1787</div>				
<h3>framebuffer驱动全篇</h3> <p>在后续的几篇里面会详细介绍如何编写一个显卡的驱动程序。 framebuffer device 在内核里面作为显卡驱动模型，许多函数和数据结构都是特定，正是这些特定的东西为我? 歪谋暗烫崩卜朔奖恪? ...</p>		 liuxd3000	2013年11月08日 18:39	 2305
<h3>framebuffer的入门介绍-实现程序分析</h3> <p>如想想对lcd屏进行操作（例如在lcd屏幕上画线，或者显示视频数据），我们就必须得了解framebuffer（帧缓冲），网上各种百度，大多都说的很官方，至少很难找到那些让人觉得很生动的描述，让我们这些出...</p>		 liuzijiang1123	2015年07月20日 19:32	 3353
<h3>Linux Framebuffer驱动剖析之一——软件需求</h3> <p>本系列文章将分析Linux Framebuffer驱动的作用（需求）、框架、接口实现和使用。按笔者一直倡导的Linux学习理念——从软件需求的角度去理解Linux，对于Linux各个子系统，我们首先要理...</p>		 yueqian_scut	2015年12月26日 14:46	 3307
<h3>Frame buffer分析 - fbcmap.c</h3> <pre>91 int fb_alloc_cmap(struct fb_cmap *cmap, int len, int transp) 92 { 93 int size = len*sizeof...</pre>		 kickxxx	2011年08月15日 18:03	 2601
<h3>fb_info结构体定义原型</h3> <pre>struct fb_info { int node; int flags; struct mutex lock; /* Lock for open/release/ioctl funcs */...</pre>		 remme123	2013年06月04日 13:58	 2916

未完待续

 leesagacious 2015年12月26日 22:56 1105

framebuffer 子系统解读

 tangcong29 2014年11月07日 18:43 1256

本文将介绍Framebuffer子系统 目标平台：TQ2 PU：s3c2440 LCD设备：3.5英寸，分辨率320X240 1. 概述 Framebuffer，中文名字是...

Linux Framebuffer驱动框架、扩展实现和使用

 Ultraman_hs 2017年02月11日 18:14 1528

一、LinuxFramebuffer的软件需求 1.针对SoC的寄存器进行编程，以支持不同的LCD屏，以使该SOC的应用场景最大化。这是硬件平台相关的需求。其对应Linux源码 arch...

Android平台截图研究 FrameBuffer(/dev/graphics/fb0) 文件内容研究！（含源码工程）

啊赛

 A1w0n 2014年07月12日 18:22 9328

android下操作Framebuffer

 neiloid 2011年11月22日 00:03 12203

一、framebuffer使用基础： 1. Linux是工作在保护模式下，所以用户态进程是无法象DOS那样使用显卡BIOS里提供的中断调用来实现直接写屏，Linux抽象出Framebuffer这个...

Android Framebuffer介绍及使用

 wx_962464 2017年09月12日 12:45 919

作者：Aaron 主页：http://www.wxtlife.com/2017/06/07/Android-framebuffer/ 欢迎订阅我的公众号 FrameBuffer...