FrameBuffer 知识基础

FrameBuffer 基础知识学习资料

搜索关键字

- *FrameBuffer* 原理
- *FrameBuffer* 应用

FrameBuffer 基础知识学习资料

FrameBuffer 基础知识网站资料

- *http://www.dzjs.net/html/qianrushixitong/2007/0516/2090.html*

什么是 *FrameBuffer*

FrameBuffer 是出现在 *Linux 2.2.xx* 内核当中的一种驱动程序接口。*Linux* 抽象出 *FrameBuffer* 这个设备来供用户态进程实现直接写屏。*FrameBuffer* 机制模仿显卡的功能，将显卡硬件结构进行抽象，可以通过 *FrameBuffer* 的读写直接对显存进行操作。用户可以将 *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、换页机制等等具体细节。这些都由 *FrameBuffer* 设备驱动来完成。

FrameBuffer 使用方法

创建 *FrameBuffer* 设备

要想系统支持 *FrameBuffer*，首先要 *Kernel* 里设置对该功能的支持，如果操作系统是 *Ubuntu*，则不需要配置 *Kernel* 选项，默认是有的，查一下 *cat /proc/device* 可以看到 *vedio* 部分 *fb* 的设备号是 *29*，若在 */dev* 下没有 *fb0* 设

备文件，则需要通过修改 *menu.lst* 来创建 *FrameBuffer*。具体方法是在 *menu.lst* 文件中找到 *Kernel* 启动项，在 *Kernel* 那行最后添加 *vga=0x317* 含义为分辨率是 *1024*768*，*16bpp*。

注意：不要进 *X windows* 来测试，应启动后直接进入字符控制台界面，否则可能播放的视频出现花屏现象，如果进了图形界面，按 *ctrl+alt+F1* 进入字符界面。

FrameBuffer 设备的应用

在应用程序中，一般通过将 *FrameBuffer* 设备映射到进程地址空间的方式使用，比如下面的程序就打开 */dev/fb0* 设备，并通过 *mmap* 系统调用进行地址映射，随后用 *memset* 将屏幕清空（这里假设显示模式是 *1024*768-8* 位色模式，线性内存模式）：

```
int fb;
unsigned char * fb_mem;
fb = open("/dev/fb0", O_RDWR);

fb_mem = mmap(NULL, 1024*768, PROT_READ|PROT_WRITE,MAP_SHARED,fb,0);
memset(fb_mem, 0, 1024*768);
```

FrameBuffer 设备还提供了若干 *ioctl* 命令，通过这些命令，可以获得显示设备的一些固定信息（比如显示内存大小）、与显示模式相关的可变信息（比如分辨率、像素结构、每扫描线的字节宽度），以及伪彩色模式下的调色板信息等。

```
struct fb_var_screeninfo fb_var
ioctl(fb, fbIOGET_VSCREENINFO, &fb_var);
int w = fb_var.xres; //显示屏幕的宽度
int h = fb_var.yres; // 显示屏幕的高度
int bpp = fb_var.bits_per_pixel; // 像素点
```

接下来，我们就可以对映射空间进行操作，以实现对设备的图像显示控制。

FrameBuffer 相关数据结构分析

FrameBuffer 设备驱动基于如下两个文件：

1) linux/include/linux/fb.h

2) linux/drivers/video/fbmem.c

下面就 *fb.h* 中的主要结构进行分析

1、fb_var_screeninfo

这个结构描述了显卡的特性

说明：__u32 代表 unsigned 无符号 32 bits 数据类型，其余类推。

这是 Linux 内核中所用到的数据类型，如果是开发用户空间(user-space)的程序，可以根据具体计算机平台的情况，用 unsigned long 等来代替

```
struct fb_var_screeninfo
{
__u32 xres;                         /* visible resolution */
__u32 yres;
__u32 xres_virtual;                 /* virtual resolution */
__u32 yres_virtual;
__u32 xoffset;                      /* offset from virtual to visible
resolution */
__u32 yoffset;

__u32 bits_per_pixel;               /* guess what */
__u32 grayscale;                    /* != 0 Gray levels instead of
colors */

struct fb_bitfield red;             /* bitfield in fb mem if true
color, */
struct fb_bitfield green;           /* else only length is significant
*/
struct fb_bitfield blue;
struct fb_bitfield transp;          /* transparency */
__u32 nonstd;                       /* != 0 Non standard pixel
format */

__u32 activate;                     /* see FB_ACTIVATE_* */

__u32 height;                       /* height of picture in mm */
__u32 width;                        /* width of picture in mm */

__u32 accel_flags;                  /* acceleration flags (hints) */

/* Timing: All values in pixclocks, except pixclock (of course) */

__u32 pixclock;                     /* pixel clock in ps (pico
seconds) */
__u32 left_margin;                  /* time from sync to picture */
__u32 right_margin;                 /* time from picture to sync */
__u32 upper_margin;                 /* time from sync to picture */
__u32 lower_margin;
```

```
__u32 hsync_len;                    /* length of horizontal sync */
__u32 vsync_len;                    /* length of vertical sync */
__u32 sync;                         /* see FB_SYNC_* */
__u32 vmode;                        /* see FB_VMODE_* */
__u32 reserved[6];                  /* Reserved for future
compatibility */
};
```