Linux 专栏收录该内容 0 订阅 63 篇文章 订阅专栏

整理了几位大牛们的图片相关的资料

framebuffer简介

帧缓冲（framebuffer）是Linux为显示设备提供的一个接口，把显存抽象后的一种设备，他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。framebuffer是LCD对应的一中HAL（硬件抽象层），提供抽象的，统一的接口操作，用户不必关心硬件层是怎么实施的。这些都是由Framebuffer设备驱动来完成的。

帧缓冲设备对应的设备文件为/dev/fb*，如果系统有多个显示卡，Linux下还可支持多个帧缓冲设备，最多可达32个，分别为/dev/fb0到/dev/fb31，而/dev/fb则为当前缺省的帧缓冲设备，通常指向/dev/fb0，在嵌入式系统中支持一个显示设备就够了。帧缓冲设备为标准字符设备，主设备号为29，次设备号则从0到31。分别对应/dev/fb0-/dev/fb31。

通过/dev/fb，应用程序的操作主要有这几种：
1．读/写（read/write）/dev/fb：相当于读/写屏幕缓冲区。
2．映射（map）操作：由于Linux工作在保护模式，每个应用程序都有自己的虚拟地址空间，在应用程序中是不能直接访问物理缓冲区地址的。而帧缓冲设备可以通过mmap()映射操作将屏幕缓冲区的物理地址映射到用户空间的一段虚拟地址上，然后用户就可以通过读写这段虚拟地址访问屏幕缓冲区，在屏幕上绘图了。
3．I/O控制：对于帧缓冲设备，对设备文件的ioctl操作可读取/设置显示设备及屏幕的参数，如分辨率，屏幕大小等相关参数。ioctl的操作是由底层的驱动程序来完成的。

在应用程序中，操作/dev/fb的一般步骤如下：
1．打开/dev/fb设备文件。
2．用ioctl操作取得当前显示屏幕的参数，根据屏幕参数可计算屏幕缓冲区的大小。
3．将屏幕缓冲区映射到用户空间。
4．映射后即可直接读写屏幕缓冲区，进行绘图和图片显示。

framebuffer相关数据结构介绍
1. fb_info结构体：帧缓冲设备中最重要的数据结构体，包括了帧缓冲设备属性和操作的完整性属性。
2. fb_ops结构体：fb_info结构体的成员变量，fb_ops为指向底层操作的函数的指针。
3.fb_var_screen和fb_fix_screen结构体：fb_var_screen记录用户可以修改的显示控制器参数，fb_fix_screen记录用户不能修改的显示控制器参数。

*.bmp文件和大多数图形文件一样，分为文件描述区（头文件信息）和图象存储区（象素数据）两部分。而头文件信息中又包含了信息区和调色板区两部分，信息区又可以细分为文件信息区和图象信息区两部分。

24位bmp格式，一位大牛用oD分析出来的

1.BMP文件的标识符，开头都是42 4D，mspaint也是靠这个判断一个文件是不是BMP文件的。
2.两个DWORD数据用来保存bmp文件的大小，一般高DWORD数据为0，因为大小超过4GB的BMP文件还是比较少见的。
3.一个DWORD数据，它指出了颜色数据开始的地址，上图中的DWORD数据是00000036，我们看下00000036地址处的数据是什么, B4 A8 90 这就是颜色数据了，由于是文件是24位的BMP图片，所以图片中的每一个像素的颜色值都用3个字节来描述，即24个二进制位。每个字节对应一种基础颜色的艳丽程度，最后三种颜色混合起来才是最终的颜色。对，你肯定想到了，RGB三基**4 A8 90 那个是红色 哪个是蓝色呢？ 经过验证，90是红色，A8是绿**4是蓝色， 这三个数据告诉显卡程序要在显示器的某个点显示一种颜色，什么样的颜色呢？90个单位的红色加上A8个单位的绿色再加上B4个单位的蓝色最后得到的那种颜色，恩，我就要那种颜色。
4.一个DWORD数据，具体意义不明，大多数情况下这个值是00 00 00 28，但是也有其它的值，LZ前几天记得至少有三个值是可以用的，LZ记得画在一张图上了，可是现在找不到了。其它的值都是不行的，程序会显示无效的图片。
5.宽度以pixel为单位
6.高度以pixel为单位
7.帧数，一般bmp文件为一帧，在动画中它是一个基础单位。

以下代码使用framebuffer显示一张bmp图片：

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <arpa/inet.h>
#include <utils/Log.h>
#include <errno.h>

//14byte文件头
typedef struct
{
    char cfType[2];//文件类型，"BM"(0x4D42)
    long cfSize;//文件大小（字节）
    long cfReserved;//保留，值为0
    long cfoffBits;//数据区相对于文件头的偏移量（字节）
}__attribute__((packed)) BITMAPFILEHEADER;
//__attribute__((packed))的作用是告诉编译器取消结构在编译过程中的优化对齐

//40byte信息头
typedef struct
{
    char ciSize[4];//BITMAPFILEHEADER所占的字节数
    long ciWidth;//宽度
    long ciHeight;//高度
    char ciPlanes[2];//目标设备的位平面数，值为1
    int ciBitCount;//每个像素的位数
    char ciCompress[4];//压缩说明
    char ciSizeImage[4];//用字节表示的图像大小，该数据必须是4的倍数
    char ciXPelsPerMeter[4];//目标设备的水平像素数/米
    char ciYPelsPerMeter[4];//目标设备的垂直像素数/米
    char ciClrUsed[4]; //位图使用调色板的颜色数
    char ciClrImportant[4]; //指定重要的颜色数，当该域的值等于颜色数时（或者等于0时），表示所有颜色都一样重要
}__attribute__((packed)) BITMAPINFOHEADER;

typedef struct
```

```c
{
    unsigned short blue;
    unsigned short green;
    unsigned short red;
    unsigned short reserved;
}__attribute__((packed)) PIXEL;//颜色模式RGB

BITMAPFILEHEADER FileHead;
BITMAPINFOHEADER InfoHead;

static char *fbp = 0;
static int xres = 0;
static int yres = 0;
static int bits_per_pixel = 0;

int show_bmp();
int show_bmp2();
int fbfd = 0;
static void fb_update(struct fb_var_screeninfo *vi)   //将要渲染的图形缓冲区的内容绘制到设备显示屏来
{
    vi->yoffset = 1;
    ioctl(fbfd, FBIOPUT_VSCREENINFO, vi);
    vi->yoffset = 0;
    ioctl(fbfd, FBIOPUT_VSCREENINFO, vi);
}

int width, height;

static int cursor_bitmpa_format_convert(char *dst,char *src){
    int i ,j ;
    char *psrc = src ;
    char *pdst = dst;
    char *p = psrc;
    int value = 0x00;

    /* 由于bmp存储是从后面往前面，所以需要倒序进行转换 */
 pdst += (width * height * 4);
    for(i=0;i<height;i++){
   p = psrc + (i+1) * width * 3;
        for(j=0;j<width;j++){
      pdst -= 4;
       p -= 3;
       pdst[0] = p[0];
       pdst[1] = p[1];
       pdst[2] = p[2];
            //pdst[3] = 0x00;

     value = *((int*)pdst);
     value = pdst[0];
           if(value == 0x00){
          pdst[3] = 0x00;
           }else{
          pdst[3] = 0xff;
           }
        }
    }

    return 0;
}

int show_bmp(char *path)
```

```
{
 FILE *fp;
    int rc;
    int line_x, line_y;
    long int location = 0, BytesPerLine = 0;
    char *bmp_buf = NULL;
    char *bmp_buf_dst = NULL;
    char * buf = NULL;
    int flen = 0;
    int ret = -1;
    int total_length = 0;

    LOGI("into show_bmp function_____________________________________________________________________________________");
    if(path == NULL)
        {
            LOGE("path Error,return");
            return -1;
        }
    LOGI("path = %s", path);
 fp  = fopen( path, "rb" );
    if(fp == NULL){
        LOGE("load > cursor file open failed");
        return -1;
    }
    /* 求解文件长度 */
    fseek(fp,0,SEEK_SET);
    fseek(fp,0,SEEK_END);
 flen  = ftell(fp);

 bmp_buf  = (char*)calloc(1,flen - 54);
    if(bmp_buf == NULL){
        LOGE("load > malloc bmp out of memory!");
        return -1;
    }

    /* 再移位到文件头部 */
    fseek(fp,0,SEEK_SET);

 rc  = fread(&FileHead, sizeof(BITMAPFILEHEADER),1, fp);
    if ( rc != 1)
    {
        LOGI("read header error!\n");
        fclose( fp );
        return( -2 );
    }

    //检测是否是bmp图像
    if (memcmp(FileHead.cfType, "BM", 2) != 0)
    {
        LOGI("it's not a BMP file\n");
        fclose( fp );
        return( -3 );
    }
 rc  = fread( (char *)&InfoHead, sizeof(BITMAPINFOHEADER),1, fp );
    if ( rc != 1)
    {
        LOGI("read infoheader error!\n");
        fclose( fp );
        return( -4 );
    }
 width = InfoHead.ciWidth;
```

```
height = InfoHead.ciHeight;
    LOGI("FileHead.cfSize =%d byte\n",FileHead.cfSize);
    LOGI("flen = %d", flen);
    LOGI("width = %d, height = %d", width, height);
total_length = width * height *3;

    LOGI("total_length = %d", total_length);
    //跳转的数据区
    fseek(fp, FileHead.cfoffBits, SEEK_SET);
    LOGI(" FileHead.cfoffBits = %d\n",  FileHead.cfoffBits);
    LOGI(" InfoHead.ciBitCount = %d\n",  InfoHead.ciBitCount);
    //每行字节数
buf = bmp_buf;
    while ((ret = fread(buf,1,total_length,fp)) >= 0) {
        if (ret == 0) {
            usleep(100);
            continue;
        }
        LOGI("ret = %d", ret);
   buf  =  ((char*) buf) + ret;
 total_length = total_length - ret;
        if(total_length == 0)break;
    }

total_length = width * height * 4;
    LOGI("total_length = %d", total_length);
bmp_buf_dst = (char*)calloc(1,total_length);
    if(bmp_buf_dst == NULL){
        LOGE("load > malloc bmp out of memory!");
        return -1;
    }

    cursor_bitmpa_format_convert(bmp_buf_dst, bmp_buf);
    memcpy(fbp,bmp_buf_dst,total_length);

    LOGI("show logo return 0");
    return 0;
}
int show_picture(int fd, char *path)
{

    struct fb_var_screeninfo vinfo;
    struct fb_fix_screeninfo finfo;
    long int screensize = 0;
    struct fb_bitfield red;
    struct fb_bitfield green;
    struct fb_bitfield blue;
    LOGI("Enter show_logo");
    //打开显示设备
retry1:
fbfd = fd;//open("/dev/graphics/fb0", O_RDWR);
    LOGI("fbfd = %d", fbfd);
    if (fbfd == -1)
    {
        LOGE("Error opening frame buffer errno=%d (%s)",
             errno, strerror(errno));
        goto retry1;
    }

    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo))
    {
```

```
223         LOGI("Error：reading fixed information.\n");
224         return -1;
225     }
226 
227     if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo))
228     {
229         LOGI("Error: reading variable information.\n");
230         return -1;
231     }
232 
233     LOGI("R:%d,G:%d,B:%d \n", vinfo.red, vinfo.green, vinfo.blue );
234 
235     LOGI("%dx%d, %dbpp\n", vinfo.xres, vinfo.yres, vinfo.bits_per_pixel );
236  xres = vinfo.xres;
237  yres = vinfo.yres;
238  bits_per_pixel = vinfo.bits_per_pixel;
239 
240     //计算屏幕的总大小（字节）
241  screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
242     LOGI("screensize=%d byte\n",screensize);
243 
244     //对象映射
245  fbp  =  (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED, fbfd, 0);
246     if ((int)fbp == -1)
247     {
248         printf("Error: failed to map framebuffer device to memory.\n");
249         return -1;
250     }
251 
252     LOGI("sizeof file header=%d\n", sizeof(BITMAPFILEHEADER));
253 
254 
255     //显示图像
256 // while(1){
257     show_bmp(path);
258     fb_update(&vinfo);
259 //}
260 
261     //删除对象映射
262     munmap(fbp, screensize);
263     //close(fbfd);
264     LOGI("Exit show_logo");
265     return 0;
266 }
```

由于bmp图象是从下至上存储的，所以我们不能进行直接顺序读取。详细的说，bmp图象存储区数据是从1078偏移字节开始。文件内第一个图象点实际上是对应图象（320*200）第200行的最左边的第一个点，而从1078开始的320个点则是图象最下面一行对应的点，之后的321个点是图象倒数第二行最左边的第一个点。这样，bmp文件最后一个字节对应的点是第一行最后边的点了。

上面程序显示的图片原来是24位深度的，代码里面将其转为32位

有几个需要注意，第一必须为bmp图片，第二，图片不能过大

注意：上面的程序只在framebuffer上显示图片，却没有删除刷新屏幕，可以使用下面的命令恢复屏幕

保存屏幕信息：dd if=/dev/fb0 of=fbfile 或： cp /dev/fb0 fbfile
恢复屏幕信息：dd if=fbfile of=/dev/fb0 或： cat fbfile > /dev/fb0

linux下framebuffer转换为bmp图片 06-09
把framebuffer的数据转换为bmp图片，超高清，windows可以直接查看图片

new_abc 专家
码龄10年 暂无认证

370	1万+	71万+	175万+	
原创	周排名	总排名	访问	等级

2万+	639	130	316	234
积分	粉丝	获赞	评论	收藏

私信 关注

vstvr： 还有一个细节，我的显示屏打印出来貌似红色和蓝色是互换的，所以还需要稍作修改，在cursor_bitmpa_format_convert中找到如下代码并修改，pdst的3个位代表三种颜色，随意换一下试试：

```
pdst[0] = p[2];
      pdst[1] = p[1];
      pdst[2] = p[0];
```

1 年前 回复

vstvr： 不愧是大佬。但是貌似下载图片的大小应该和显示屏大小一致，如果下载了一个小图片然后用在线软件整成大图，显示的时候就会出现多重影分身的效果。建议如果屏幕是1280*800，最好下载图片为1280*853（32位），然后在showbmp函数的末端改成

```
width = InfoHead.ciWidth;
 height = InfoHead.ciHeight-53
```

1 年前 回复

嵌入式linux操作framebuffer显示bmp图片 小飞的博客 2511
编译后拷贝进开发板即可使用 使用方法 ./fb_show_bmp test.bmp 显示的图片由参数指定，上面指令中test.bmp为测试用的bmp格式的图片 效果 源码说明 …

linux bmp图片应用编程,在Framebuffer下编程显示BMP图象 分享 最新发布 weixin_29238147的博客 36
http://yaos.blog.sohu.com/20240931.html今天看别人的代码，知道可以使控制台进入图形模式，这样SHELL程序的显示就不会影响BMP图像的显示了。…

利用framebuffer，命令行显示图片 weixin_34411563的博客 208
上代码 import fcntl import struct import mmap import contextlib import os import time import numpy as np import cv2 src = cv2.imread('./1.bmp', cv2.IMR…

通过写framebuffer显示BMP图片 yuyin86的专栏 1448
#include #include #include #include #include #include #include #include #include //14byte文件头typedef struct{char cfType[2];//文件类型，"

c代码实现framebuffer显示jpeg图片 12-22
c语言实现的framebuffer显示jpeg图片，并且支持扭动显示 显示曲线是sin曲线

基于framebuffer显示图片(bmp & png)和汉字 杨善锦技术专栏 3644
章节 1 功能描述 2 图片显示，字体显示接口 3 使用实例 功能描述图片显示: 支持位置设置，支持BMP & PNG格式的图片，程序通过读取bmp&…

framebuffer BMP图像显示 11-22
在嵌入式Linux系统上操作framebuffer显示BMP图像

framebuffer显示图片 03-28
在terminal里写framebuffer显示一张32位/24位/16位jpg图片

FrameBuffer系列 之 显示图片 土著部落 5426
摘自：http://blog.csdn.net/luxiaoxun/article/details/7622988 #include #include #include #include #include #include #include #include //14byte…

framebuffer显示JPEG图片 liu0808的专栏 2086
framebuffer显示JPEG图片 2011-05-03 20:14:45 分类： 嵌入式 转自http://www.linuxsense.org/archives/281.html http://kb.cnblogs.com/a/1526327/ apt-…

framebuffer显示JPEG或BMP图片 10-01
framebuffer中显示JPEG或BMP图片，文档中有源码，可以直接编译运行

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

css 实现图片宽度自适屏幕，高度与宽度成固定比例

有关css伪元素before after

table-cell布局

2019年 5篇　2018年 9篇
2017年 12篇　2016年 101篇
2015年 162篇　2014年 77篇
2013年 72篇　2012年 145篇

arm平台framebuffer 显示png图片　12-22
arm平台下，交叉编译，实现png图片的解析 通过framebuffer的操作 显示png图片到lcd上

linux下bmp图像显示　07-19
本程序中用比简单的方法来进行bmp图片的读取，并用linux 下的framebuffer来显示到屏幕上

linux图像显示（二）bmp图片　JT同学的博客　3805
linux图像显示 linux图像显示（一）framebuffer操作 linux图像显示（二）bmp图片 linux图像显示（三）使用libjpg处理jpg图片 linux图像显示（四）使用libp…

linux下的framebuffer显示图片 分类： arm-linu...　weixin_30781775的博客　143
void showbmp2() { int x,y; unsigned char *p; int index=0; struct fb_var_screeninfo vinfo; struct fb_fix_screeninfo finfo; struct fb_bitfield red; ...


new_abc　关注　4　2　16　专栏目录

举报