

原创

干燥剂007860

2021-04-30 13:31:48


536

收藏 12

版权

分类专栏：Linux

文章标签：嵌入式linux

Linux 专栏收录该内容

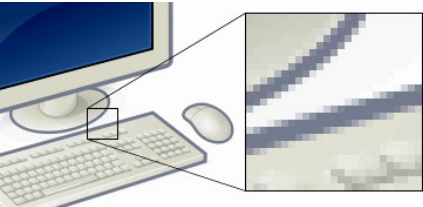
0 订阅

7 篇文章

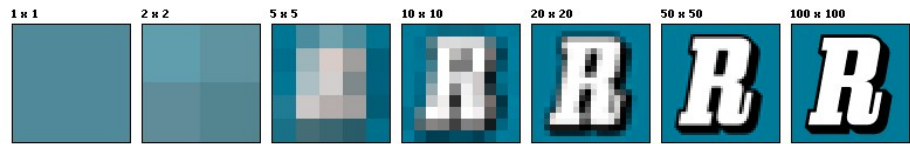
订阅专栏

1. 液晶屏的基本概念

- 像素：
屏幕上显示颜色的最小单位，英文叫 pixel。注意，位图（如jpg、bmp等格式的常见图片）也是由一个个的像素点构成的，跟屏幕的像素点的概念一样。原理上讲，将一张位图显示到屏幕上，就是将图片上的像素点一个个复制到屏幕像素点上。



- 分辨率：
 - 宽、高两个维度上的像素点数目。
 - 分辨率越高，所需要的显存越大。



- 色深：
 - 每个像素所对应的内存字节数，一般有8位、16位、24位或32位
 - GEC6818开发板的屏幕的色深是32位的
 - 32位色深的屏幕一般被称为真彩屏，或1600万色屏。

色深决定了一个像素点所能表达的颜色丰富程度，色深越大，色彩表现力越强。

2. 内存映射基本原理

虽然LCD设备本质上也可以看作是一个文件，在文件系统中有其对应的设备节点，可以像普通文件一样对其进行读写操作（read/write），但由于对字符设备的读写操作是以字节流的方式进行的，因此除非操作的图像尺寸刚好与屏幕尺寸完全一致，如下图所示，图片的宽高与LCD的宽高完全一致，否则将会画面会乱。

目录

- 液晶屏的基本概念
- 内存映射基本原理
- 屏幕参数设定

「课堂练习1」

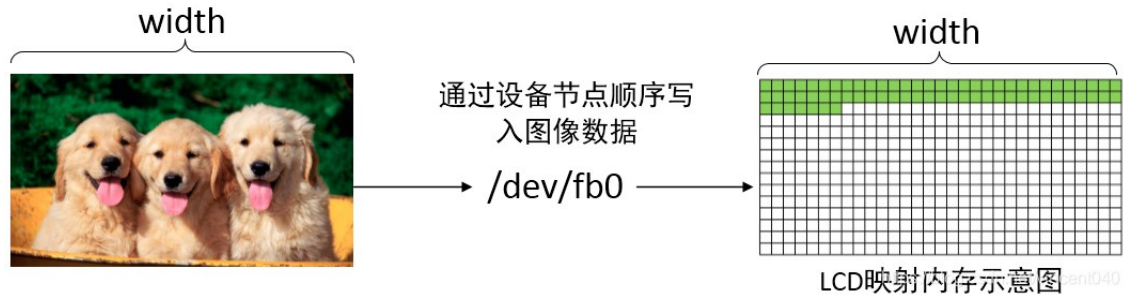
4. 多缓冲机制

节选自：粤嵌-嵌入式课堂笔记

联系人：18028569040（曾小美老...）

分类专栏

	图片库	1篇
	Qt程序设计	
	技术栈	89篇
	C语言	10篇
	数据结构与算法	5篇
	C++基础语法	7篇
	Linux	7篇
	SQLite	1篇
	视频教程	2篇

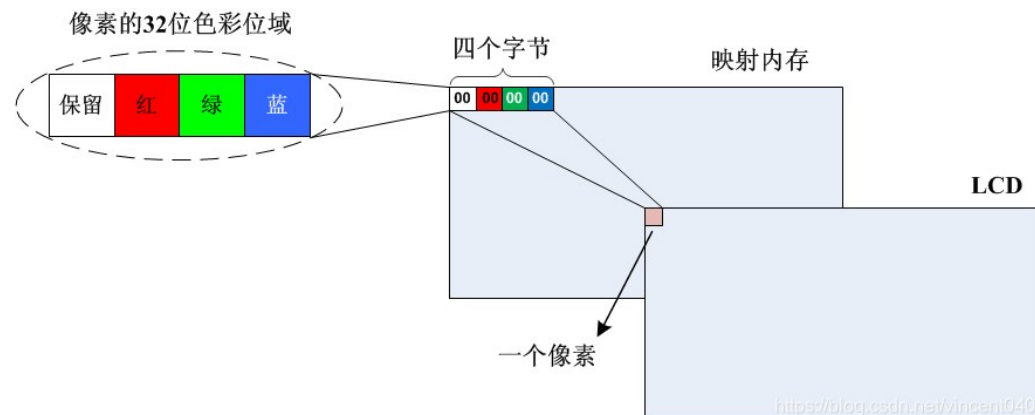


以下是一段直接写设备节点的“不好”的示例代码：

```
1 void bad_display()
2 {
3     // 打开LCD设备
4     int lcd = open("/dev/fb0", O_RDWR);
5
6     // 从JPG图片中获取ARGB数据
7     char *argbbuf;
8     int argbsize;
9     argbsize = jpg2rgb("dogs.jpg", &argbbuf);
10
11     // 将RGB数据直接线性灌入LCD设备节点
12     write(lcd, argbbuf, argbsize);
13
14     // ...
15 }
```

像上述代码这样，直接将数据通过设备节点 /dev/fb0 写入的话，这些数据会自动地从LCD映射内存的入口处（对应LCD屏幕的左上角）开始呈现，并且会以线性的字节流形式逐个字节往后填充，除非图像尺寸与显示器刚好完全一致，否则显示是失败的。

一般而言，图像的尺寸大小是随机的，因此更方便的做法是为LCD做内存映射，将屏幕的每一个像素点跟映射内存一一对应，而映射内存可以是二维数组，因此就可以非常方便地通过操作二维数组中的任意元素，来操作屏幕中的任意像素点了。这里的映射内存，有时被称为显存。



如上图所示，将一块内存与LCD的像素——对应：

1. LCD上面显示的图像色彩，由其对应的内存的数据决定
2. 映射内存的大小至少得等于LCD的真实尺寸大小
3. 映射内存的大小可以大于LCD的真实尺寸，有利于优化动态画面（视频）体验

下面是屏幕显示为红色的示例代码：

```
1  #include <stdio.h>
2  #include <sys/mman.h>
3  #include <string.h>
4  #include <fcntl.h>
5
6  int main()
7  {
8      // 打开液晶屏文件
9      int lcd = open("/dev/fb0", O_RDWR);
10
11     // 给LCD设备映射一块内存（或称显存）
12     char *p = mmap(NULL, 800*480*4, PROT_WRITE,
13                     MAP_SHARED, lcd, 0);
14
15     // 通过映射内存，将LCD屏幕的每一个像素点涂成红色
16     int red = 0xFF0000;
17     for(int i=0; i<800*480; i++)
18         memcpy(p+i*4, &red, 4);
19
20     // 解除映射
21     munmap(p, 800*480*4);
22     return 0;
23 }
```

注意，上述代码存在诸多假设，比如屏幕的尺寸是800×480、屏幕色深是4个字节、每个像素内部的颜色分量是ARGB等等，这些信息都是“生搬硬凑”的，只能适用于某一款特定的LCD屏，如果屏幕的这些参数变了，上述代码就无法正常运行了，要想让程序在其他规格尺寸的屏幕下也能正常工作，就得让程序自动获取这些硬件参数信息。

3. 屏幕参数设定

首先明确，屏幕的硬件参数，都是由硬件驱动工程师，根据硬件数据手册和内核的相关规定，填入某个固定的地方的，然后再由应用开发工程师，使用特定的函数接口，将这些特定的信息读出来。

对于GEC6818开发板而言，上述所谓“某个固定的地方”，指的是如下这些重要的结构体（节选）：

```
1  struct fb_fix_screeninfo
2  {
3      char id[16];           /* identification string eg "TT Builtin" */
4      unsigned long smem_start; /* Start of frame buffer mem */
5                               /* (physical address) */
6      __u32 smem_len;        /* Length of frame buffer mem */
7      __u32 type;            /* see FB_TYPE_* */
8      __u32 type_aux;        /* Interleave for interleaved Planes */
9      __u32 visual;          /* see FB_VISUAL_* */
10     __u16 xpanstep;         /* zero if no hardware panning */
11     __u16 ypanstep;         /* zero if no hardware panning */
12     __u16 ywrapstep;        /* zero if no hardware ywrap */
13     __u32 line_length;      /* Length of a Line in bytes */
14 }
```

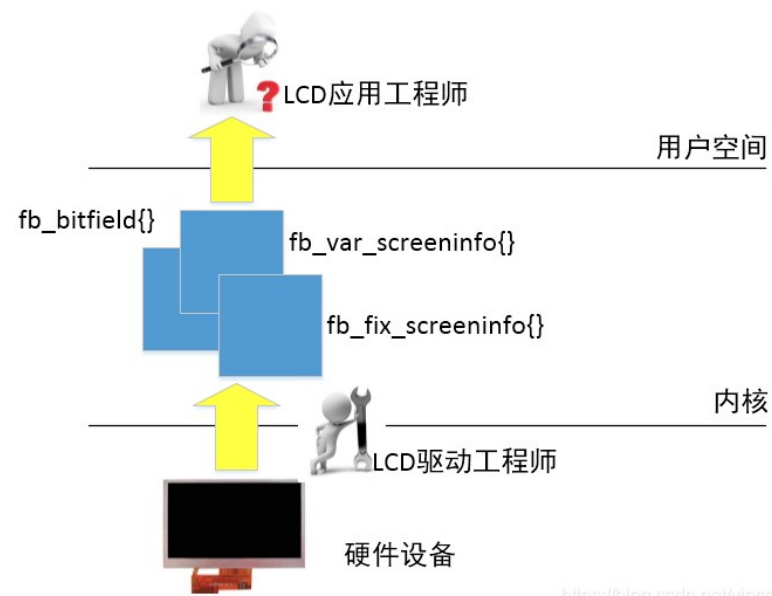
```

14  ...
15  ...
16  };
17
18  struct fb_var_screeninfo
19  {
20      __u32 xres;          /* 可见区宽度(单位:像素) */
21      __u32 yres;          /* 可见区高度(单位:像素) */
22      __u32 xres_virtual;  /* 虚拟区宽度(单位:像素) */
23      __u32 yres_virtual;  /* 虚拟区高度(单位:像素) */
24      __u32 xoffset;       /* 虚拟区到可见区x轴偏移量 */
25      __u32 yoffset;       /* 虚拟区到可见区y轴偏移量 */
26
27      __u32 bits_per_pixel; /* 色深 */
28
29      // 像素内颜色结构
30      struct fb_bitfield red;  // 红色
31      struct fb_bitfield green; // 绿色
32      struct fb_bitfield blue; // 蓝色
33      struct fb_bitfield transp; // 透明度
34      ...
35      ...
36  };
37
38  struct fb_bitfield
39  {
40      __u32 offset; /* 颜色在像素内偏移量 */
41      __u32 length; /* 颜色占用数位长度 */
42      ...
43      ...
44  };

```

上述结构体的具体定义在系统的如下路径中：

```
1 | /usr/include/linux/fb.h
```



如上图所示，如果板卡已经具备LCD的驱动程序，那么应用程序就可以通过 `ioctl()` 来检索LCD的硬件参数信息。以粤嵌GEC6818开发板配套的群创AT070TN92-7英寸液晶显示屏为例，具体代码如下：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <stdbool.h>
5  #include <errno.h>
6
7  #include <sys/types.h>
8  #include <sys/mman.h>
9  #include <sys/ioctl.h>
10 #include <linux/fb.h>
11 #include <fcntl.h>
12
13 int lcd;
14 struct fb_fix_screeninfo fixinfo; // 固定属性
15 struct fb_var_screeninfo varinfo; // 可变属性
16
17 void get_fixinfo()
18 {
19     if(ioctl(lcd, FBIOGET_FSCREENINFO, &fixinfo) != 0)
20     {
21         perror("获取LCD设备固定属性信息失败");
22         return;
23     }
24 }
25
26
27 void get_varinfo()
28 {
29     if(ioctl(lcd, FBIOGET_VSCREENINFO, &varinfo) != 0)
30     {
31         perror("获取LCD设备可变属性信息失败");
32         return;
33     }
34 }
35
36 void show_info()
37 {
38     // 获取LCD设备硬件fix属性
39     get_fixinfo();
40     printf("\n获取LCD设备固定属性信息成功：\n");
41     printf("[ID]: %s\n", fixinfo.id);
42     printf("[FB类型]: ");
43     switch(fixinfo.type)
44     {
45         case FB_TYPE_PACKED_PIXELS:    printf("组合像素\n");break;
46         case FB_TYPE_PLANES:           printf("非交错图层\n");break;
47         case FB_TYPE_INTERLEAVED_PLANES: printf("交错图层\n");break;
48         case FB_TYPE_TEXT:             printf("文本或属性\n");break;
49         case FB_TYPE_VGA_PLANES:       printf("EGA/VGA图层\n");break;
50     }
51     printf("[FB视觉]: ");
52     switch(fixinfo.visual)
53     {
54         case FB_VISUAL_MONO01:         printf("灰度, 1=黑;0=白\n");break;
```

```

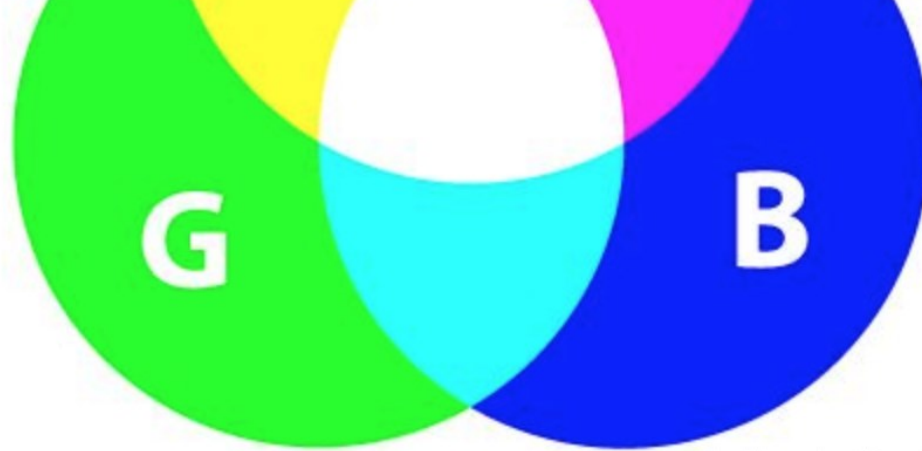
55     case FB_VISUAL_MONO10:           printf("灰度. 0=黑;1=白\n");break;
56     case FB_VISUAL_TRUECOLOR:        printf("真彩色\n");break;
57     case FB_VISUAL_PSEUDOCOLOR:      printf("伪彩色\n");break;
58     case FB_VISUAL_DIRECTCOLOR:       printf("直接彩色\n");break;
59     case FB_VISUAL_STATIC_PSEUDOCOLOR: printf("只读伪彩色\n");break;
60 }
61 printf("[行宽]: %d 字节\n", fixinfo.line_length);
62
63 // 获取LCD设备硬件var属性
64 get_varinfo();
65 printf("\n获取LCD设备可变属性信息成功: \n");
66 printf("[可见分辨率]: %d×%d\n", varinfo.xres, varinfo.yres);
67 printf("[虚拟分辨率]: %d×%d\n", varinfo.xres_virtual, varinfo.yres_virtual);
68 printf("[从虚拟区到可见区偏移量]: (%d,%d)\n", varinfo.xoffset, varinfo.yoffset);
69 printf("[色深]: %d bits\n", varinfo.bits_per_pixel);
70 printf("[像素内颜色结构]:\n");
71 printf("  [红] 偏移量:%d, 长度:%d bits\n", varinfo.red.offset, varinfo.red.length);
72 printf("  [绿] 偏移量:%d, 长度:%d bits\n", varinfo.green.offset, varinfo.green.length);
73 printf("  [蓝] 偏移量:%d, 长度:%d bits\n", varinfo.blue.offset, varinfo.blue.length);
74 printf("  [透明度] 偏移量:%d, 长度:%d bits\n", varinfo.transp.offset, varinfo.transp.length);
75 printf("\n");
76 }
77
78 int main()
79 {
80     lcd = open("/dev/fb0", O_RDWR);
81     if(lcd == -1)
82     {
83         perror("打开 /dev/fb0 失败");
84         exit(0);
85     }
86
87     // 显示LCD设备属性信息
88     show_info();
89
90     return 0;
91 }

```

「课堂练习1」

根据以上示例代码，采用自动获取屏幕硬件参数的方式，在开发板上轮流显示红绿蓝三原色。





<https://blog.csdn.net/vinceni040>

4. 多缓冲机制

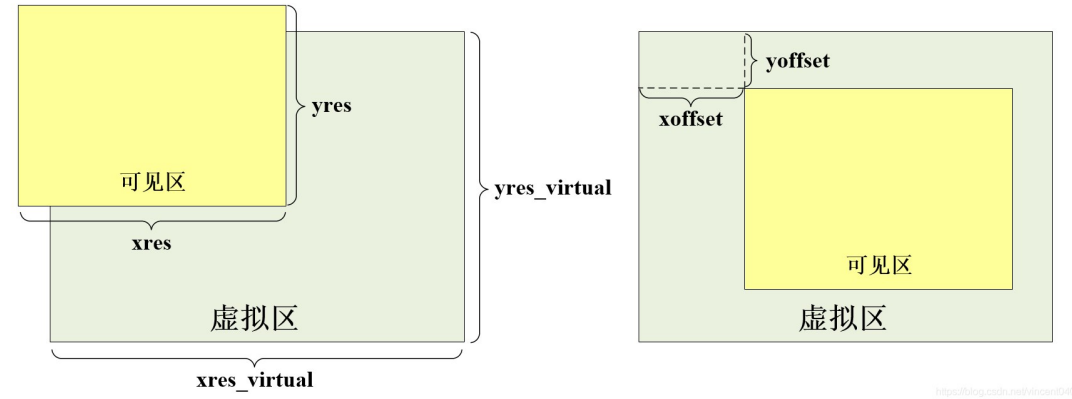
仔细观察上述显示单色的程序运行效果，会发现屏幕上的颜色不是一瞬间整体显示的，而是有一个很明显的从上到下刷屏的过程，这实际上是由于我们是一个个像素点从左到右，从上到下刷屏导致的，如果不是速度比较快，我们将会看到屏幕上的点是一个个亮起来的，而不是整屏统一更新，这显然不是最佳的体验。



解决这个问题，可以采用多缓冲的办法，首先要搞明白所谓可见区和虚拟区的关系：

1. 可见区、虚拟区都是内存区域，可见区是虚拟区的一部分，因此可见区尺寸至少等于虚拟区。

2. 一般而言，可见区尺寸就是屏幕尺寸，比如800×480；而虚拟区是显示设备能支持的显存大小，比如800×480、800×960等。
3. 为了提高画面体验，一般先在不可见区操作显存数据，然后在调整可见区位置，使得图像“瞬间”呈现，避免闪屏。



下面以示例代码的形式，来分析如何使用多缓冲机制提高画面体验。

- 1. 设定虚拟区

```
1 #include <stdio.h>
2 #include <sys/mman.h>
3 #include <sys/ioctl.h>
4 #include <string.h>
5 #include <fcntl.h>
6 #include <linux/fb.h>
7
8 int main()
9 {
10     // 打开LCD设备
11     int lcd = open("/dev/fb0", O_RDWR|O_EXCL);
12
13     struct fb_var_screeninfo vinfo; // 显卡设备的可变属性结构体
14     ioctl(lcd, FBIOGET_VSCREENINFO, &vinfo); // 获取可变属性
15
16     // 获得当前显卡所支持的虚拟区显存大小
17     unsigned long VWIDTH = vinfo.xres_virtual;
18     unsigned long VHEIGHT = vinfo.yres_virtual;
19     unsigned long BPP = vinfo.bits_per_pixel;
20
21     printf("虚拟区显存大小为: %d×%d\n", VWIDTH, VHEIGHT);
22
23     // 申请一块虚拟区大小的映射内存
24     char *p = mmap(NULL, VWIDTH * VHEIGHT * BPP/8,
25                     PROT_READ|PROT_WRITE,
26                     MAP_SHARED, lcd, 0);
27     if(p != MAP_FAILED)
28     {
29         printf("申请显存成功\n");
30     }
31 }
```

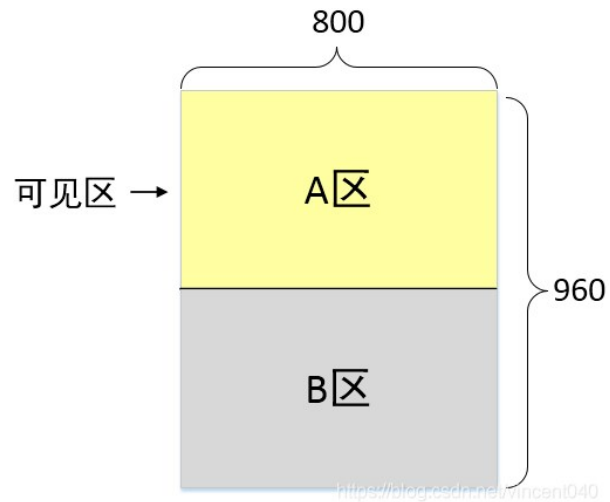

在开发板运行结果：

```
1 [root@GEC6818 ~]# ./a.out
2 虚拟区显存大小为: 800x1440
3 申请显存成功
4 [root@GEC6818 ~]#
```

从上述执行结果来看，粤嵌GEC6818开发板配套的群创AT070TN92-7英寸液晶显示屏支持三倍与屏幕尺寸的虚拟显存的设定。当然，在实际设定的时候，不一定要三倍，也可以是两倍大小，比如800×960。

- 2. 显示A区，但在B区作画

为了方便讨论，假设设定两倍屏幕尺寸的虚拟区内存，上半部分为A区，下半部分为B区。如下图所示：



将A区设定为可见区，代码如下：

```
1 struct fb_var_screeninfo vinfo; // 显卡设备的可变属性结构体
2 ioctl(lcd, FBIOGET_VSCREENINFO, &vinfo); // 获取可变属性
3
4 // 获得当前显卡所支持的虚拟区显存大小
5 unsigned long width = vinfo.xres;
6 unsigned long height = vinfo.yres;
7 unsigned long bpp = vinfo.bits_per_pixel;
8 unsigned long screen_size = width * height * bpp/8;
9
10 // 申请一块两倍与屏幕的映射内存
11 char *p = mmap(NULL, 2 * screen_size,
12                PROT_READ|PROT_WRITE,
13                MAP_SHARED, lcd, 0);
14
15 // 将可见区设定为A区
16 vinfo.xoffset = 0;
17 vinfo.yoffset = 0;
18 ioctl(lcd, FBIOPAN_DISPLAY, &vinfo);
19
```

```

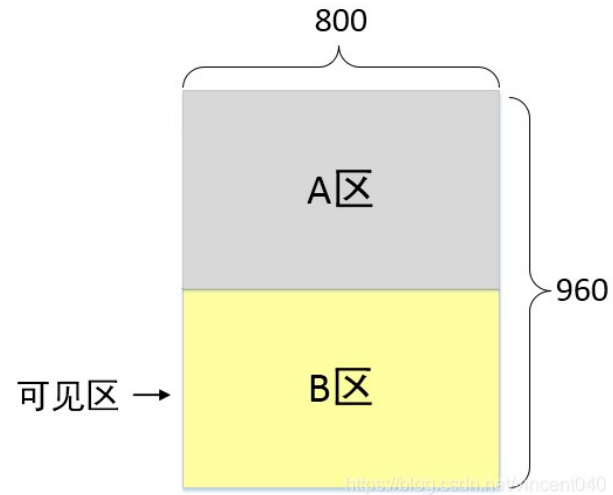
20 // 在B区绘图
21 int red = 0x00FF0000;
22 for(int i=0; i<width*height; i++)
23     memcpy(p+screen_size+i*4, &red, 4);

```

执行上述代码，会发现虽然在B区已经填充了某些图像数据，但是屏幕上没有出现任何反应。

- **3. 将可见区设定为B区，瞬间出现画面，避免了“闪屏”**

为了方便讨论，假设设定两倍屏幕尺寸的虚拟区内存，上半部分为A区，下半部分为B区。如下图所示：



将B区设定为可见区，代码如下：

```

1 | vinfo.xoffset = 0;
2 | vinfo.yoffset = 480;
3 | ioctl(lcd, FBIOPAN_DISPLAY, &vinfo);

```

容易想到，只要交替地改变可见区，使得填充数据的过程对用户不可见，等到数据填充完毕，再通过以上代码瞬间调整可见区区域，用户就能感受到画面流程呈现的体验，避免尴尬的闪屏。

下面是完整的使用“双缓冲”机制交替呈现红绿蓝的代码及演示效果图。

```

1 | #include <stdio.h>
2 | #include <unistd.h>
3 | #include <sys/mman.h>
4 | #include <sys/ioctl.h>
5 | #include <string.h>
6 | #include <fcntl.h>
7 | #include <linux/fb.h>
8 |
9 | int main()
10 | {
11 |     // 打开LCD设备

```

```

12 int lcd = open("/dev/fb0", O_RDWR|O_EXCL);
13
14 struct fb_var_screeninfo vinfo; // 显卡设备的可变属性结构体
15 ioctl(lcd, FBIOGET_VSCREENINFO, &vinfo); // 获取可变属性
16
17 // 获得当前显卡所支持的虚拟区显存大小
18 unsigned long width = vinfo.xres;
19 unsigned long height = vinfo.yres;
20 unsigned long bpp = vinfo.bits_per_pixel;
21 unsigned long screen_size = width * height * bpp/8;
22
23 // 申请一块两倍与屏幕的映射内存
24 char *p = mmap(NULL, 2 * screen_size,
25                PROT_READ|PROT_WRITE,
26                MAP_SHARED, lcd, 0);
27
28 bzero(p, 2*screen_size);
29
30 // 将起始可见区设定为B区
31 vinfo.xoffset = 0;
32 vinfo.yoffset = 480;
33 ioctl(lcd, FBIOPAN_DISPLAY, &vinfo);
34
35 int colors[] = {0x00FF0000, 0x0000FF00, 0x000000FF};
36 for(int k=0,n=0;; n++,k++,k%=3)
37 {
38     for(int i=0; i<width*height; i++)
39         memcpy(p+ screen_size*(n%2) +i*4, &colors[k], 4);
40
41     vinfo.xoffset = 0;
42     vinfo.yoffset = 480*(n%2);
43     ioctl(lcd, FBIOPAN_DISPLAY, &vinfo);
44
45     sleep(1);
46 }
47 }

```





干燥剂007860

码龄11年
  暂无认证

152	4w+	1w+	34w+	
原创	周排名	总排名	访问	等级

4978	866	245	140	369
积分	粉丝	获赞	评论	收藏








私信

关注

搜博主文章 

热门文章

- 漫谈C语言指针入门 46711
- Linux基础（Ubuntu网络配置） 18485
- C语言（跳转语句中的流氓） 11116
- C语言（浮点精度） 9728
- C语言（回调函数） 9609

最新评论

漫谈C语言指针入门
betterlx: 我觉得挺好玩

C语言（浮点精度）
是诺诺呀: 所以下面的代码呢/

漫谈C语言指针入门
小码农下士: 来讲讲*&是啥

C语言（scanf函数工作原理）
你好嵌入式: 要是大佬再加个简单例子就好了

C语言（回调函数）
范JJ: 三连了！看其他很模糊，看博主的很清晰

您愿意向朋友推荐“博客详情页”吗？

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

干燥剂007860

关注

👍 3

💬 0

★ 12

专栏目录

【立即申报】中国科协开源评选

广告关闭

让优秀的开源更有价值，欢迎申报中国科协主办“科创中国”开源创新榜单评选

C++双缓冲技术 图像闪烁

10-17

利用双缓冲技术解决C++画图闪烁问题，例如做一个转动的太极图

请发表有价值的评论， 博客评论不欢迎灌水，良好的社区氛围需大家一起维护。

抢沙发

评论

Linux双缓冲的用法,c – 使用linux framebuffer进行适当的双缓冲

10-21

任何帮助都会对这个主题非常感激! 由于请求,这里是我想要开始工作的代码: fb0 = open("/dev/fb0", O_RDWR); if(fb0 == 0) error("Could not open frameb...

Linux framebuffer双缓冲防止闪烁

Netfilter.iptables/OpenVPN/TCP guard-(4729)

昨天写了一篇文章：使用Linux Framebuffer绘制32位真彩图形： https://blog.csdn.net/dog250/article/details/90113737 并发了朋友圈表示这件事结束了...

Linux 下framebuffer 帧缓冲的使用

diyudong4681的博客 640

framebuffer 帧缓冲 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...

linux fb双缓存,framebuffer 双缓冲问题

weixin_39573822的博客 124

刚刚接触linux，不太了解，可能会问一些比较低级的问题，请包涵。我看了网上的一些Framebuffer的资料，有提到通过设置yoffset的值来实现双缓冲的。...

Android图形系统的分析与移植–七、双缓冲framebuffer的实现

七夜雪主 8568

1 实现原理在基本的FrameBuffer已经实现的基础上，需要实现的是与Android原本模拟器所使用的goldfish FrameBuffer之间的区别。比较一下不难发现...

双缓冲 framebuffer 切换

molibaoBei90的专栏 7008

双缓冲机制 最早解释多缓冲区如何工作的方式，是通过一个现实生活中的实例来解释的。在一个阳光明媚的日子，你想将水池里的水换掉，而又找不到水...

DirectFB学习之修改FrameBuffer驱动支持双缓冲

jxgz_leo的博客 3173

DirectFB学习之修改FrameBuffer驱动支持双缓冲 折腾了一段时间基于nuc972平台的2D硬件加速驱动终于在DirectFB上跑起来了，但是我发现只要我想在...

FrameBuffer初探

KofuXu 1102

1，在前面Android系统的开机画面显示过程分析一文中提到，Linux内核在启动的过程中会创建一个类别和名称分别为"graphics"和"fb0"的设备。这样就...

Linux双缓冲的用法,使用linux framebuffer进行适当的双缓冲

weixin_35606769的博客 93

我想知道如何正确地双重缓冲帧缓冲以避免撕裂。我已经对这个主题进行了大量研究，似乎找不到任何东西。我尝试过FBIOWAITFORVSYNC。但根据这...

双缓冲-- double framebuffer

weixin_30409849的博客 108

双缓冲:在计算机上的动画与实际动画有些不同:实际的动画都是先画好了，播放的时候直接拿出来显示就行。计算机动画则是画一张，就拿出来一张，...

双缓冲(Double Buffer)原理和使用 热门推荐

业精于勤荒于嬉，行成于思毁于随 3万+

一、双缓冲作用 双缓冲甚至是多缓冲，在许多情况下都很有用。一般需要使用双缓冲区的地方都是由生产者消费者供需不一致所造成的。...

双缓冲机制

farmwang的专栏 6925

最早解释多缓冲区如何工作的方式，是通过一个现实生活中的实例来解释的。在一个阳光明媚的日子，你想将水池里的水换掉，而又找不到水管的时候，...

Qt5双缓冲机制

hjhonw的博客 3170

所谓双缓冲机制，即在绘制控件时，首先要将绘制的内容绘制在一个图片中，再将图片一次性绘制到控件上。在早期的Qt版本中，若直接在控件上进行绘...

Qt5 双缓冲机制

MZCY的博客 881

双缓冲机制



C语言中的二等公民在C++是如何翻身的？	
Linux动态库的参数化	
数据结构基本概念	
2021年 4篇	2020年 2篇
2018年 3篇	2017年 8篇
2016年 89篇	2015年 3篇
2014年 4篇	2013年 8篇
2012年 5篇	2011年 12篇
2010年 15篇	

【踩坑】并行线程消息通信——内存双缓冲存储区的实现(单向数据流动)

Trouble_provider的博客 395

近期，由于需要将运行度差异较大的三个功能模块，整合到一个系统中，此处不如将三个模块分别记为,它们的关系用图表示如下： 其中，A,B,C三个模块...

c语言双缓冲缓冲机制 最新发布

weixin_45695686的博客 145

这里写自定义目录标题 1：code: 代码转载于：（部分错误已经修改） 版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文...

闪屏情况处理

知识空间 1377

surfaceflinger中绘制图像，会把每一层的图像进行叠加，处理后的图像会显示在屏幕上。如果是某一层的图像出错导致，就必须把surfaceflinger每一层的...

©2021 CSDN 皮肤主题: 大白 设计师:CSDN官方博客 返回首页

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉 出版物许可证 营业执照

