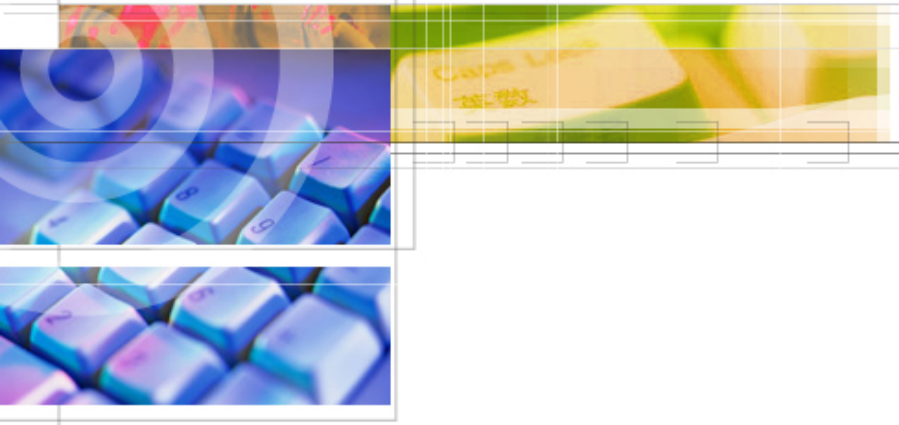


图形界面设计--Framebuffer

华中科技大学电信学院
鄢舒
E-mail: yan0shu@gmail.com

1.1 帧缓冲设备(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。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。这些都是由Framebuffer设备驱动来完成的。
- 帧缓冲驱动的应用广泛，在Linux的桌面系统中，Xwindow服务器就是利用帧缓冲进行窗口的绘制，尤其是通过帧缓冲还可显示汉字点阵。

1.2 Framebuffer编程模型

- Linux Framebuffer 本质上只是提供了对图形设备的硬件抽象，在开发者看来，Framebuffer 是一块显示缓存，往显示缓存中写入特定格式的数据就意味着向屏幕输出内容。所以说Framebuffer就是一块白板。例如对于初始化为16 位色的Framebuffer 来说， Framebuffer中的两个字节代表屏幕上一个点，从上到下，从左至右，屏幕位置与内存地址是顺序的线性关系。
- 帧缓存可以在系统存储器(内存)的任意位置，视频控制器通过访问帧缓存来刷新屏幕。 帧缓存也叫刷新缓存Framebuffer或refresh buffer， 这里的帧(frame)是指整个屏幕范围。
- 帧缓存有个地址，是在内存里。我们通过不停的向Framebuffer中写入数据， 显示控制器就自动的从Framebuffer中取数据并显示出来。全部的图形都共享内存中同一个帧缓存。

1.3 Linux的Framebuffer设备

- 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。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。分别对应从/dev/fb0到/dev/fb31。
- /dev/fb为当前缺省的帧缓冲设备，通常指向/dev/fb0。当然在嵌入式系统中支持一个显示设备就够了。
- 用#cat /dev/fb0 > screenshot将屏幕内存导入一个文件，恢复刚才的屏幕截图则可使用：

#cat screenshot > /dev/fb0。

1.4 对Framebuffer设备的操作

- 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。例如用 cp /dev/fb0 tmp命令可将当前屏幕的内容拷贝到一个文件中，而命令cp tmp > /dev/fb0 则将图形文件tmp显示在屏幕上。
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。为此， Linux在文件操作file_operations结构中提供了mmap函数，可将文件的内容映射到用户空间。对于帧缓冲设备，则可通过映射操作，可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中，之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。实际上，使用帧缓冲设备的应用程序都是通过映射操作来显示图形的。由于映射操作都是由内核来完成，下面我们将看到，帧缓冲驱动留给开发人员的工作并不多。
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，显示颜色数，屏幕大小等等。ioctl的操作是由底层的驱动程序来完成的。

1.5 操作Framebuffer的步骤

1. 打开/dev/fb设备文件。
2. 用ioctl操作取得/改变当前显示屏幕的参数，如屏幕分辨率，每个像素点的比特数。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. 映射后就可以直接读写屏幕缓冲区，进行绘图和图片显示了。

1.6 Framebuffer设备的信息结构(1/5)

- 从Framebuffer设备取回的信息，有两个结构包含着我们需要的信息。第一个包含固定的屏幕信息，这部分是由硬件和驱动的能力决定的；第二个包含着可变的屏幕信息，这部分是由硬件的当前状态决定的，可以由用户空间的程序调用ioctl()来改变。
- 分别是struct fb_fix_screeninfo和struct fb_var_screeninfo。

1.6 Framebuffer设备的信息结构(2/5)

```
struct fb_fix_screeninfo {
    char id[16];  /* identification string eg "TT Builtin" */
    unsigned long smem_start;  /* Start of frame buffer mem(physical ad
    dress) */
    __u32 smem_len; /* Length of frame buffer mem */
    __u32 type; /* see FB_TYPE_* */
    __u32 type_aux; /* Interleave for interleaved Planes */
    __u32 visual;  /* see FB_VISUAL_* */
    __u16 xpanstep;  /* zero if no hardware panning */
    __u16 ypanstep;  /* zero if no hardware panning */
    __u16 ywrapstep; /* zero if no hardware ywrap */
    __u32 line_length; /* length of a line in bytes */
    unsigned long mmio_start; /*Start of Memory Mapped I/O(physical add
    ress) */
    __u32 mmio_len;  /* Length of Memory Mapped I/O */
    __u32 accel;  /* Type of acceleration available */
    __u16 reserved[3];  /* Reserved for future compatibility */
};
```

- 在这里非常重要的域是smem_len和line-length。smem-len告诉我们framebuffer设备的大小，第二个域告诉我们指针应该前进多少字节去得到下一行的数据。

1.6 Framebuffer设备的信息结构(3/5)

```
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 Graylevels 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*/};
```

```
struct fb_bitfield {
__u32 offset; /* beginning of bitfield */
__u32 length; /* length of bitfield */
__u32 msb_right; /* != 0 Most significant bit is right */
};
```

1.6 Framebuffer设备的信息结构(4/5)

- 第二个结构的前几个成员决定了分辨率。xres和yres是在屏幕上可见的实际分辨率，在通常的vga模式将为640和480。*res-virtual决定了构建屏幕时视频卡读取屏幕内存的方式。例如当实际的垂直分辨率为400，虚拟分辨率可以是800。这意味着800行的数据被保存在了屏幕内存区中。因为只有400行可以被显示，决定从那一行开始显示就是你的事了。这个可以通过设置*offset来实现。给yoffset赋0将显示前400行，赋35将显示第36行到第435行，如此重复。这个功能在许多情形下非常方便实用。
- 它可以用来做双缓冲。双缓冲就是你的程序分配了可以填充两个屏幕的内存。将offset设为0，将显示前400行，同时可以秘密的在400行到799行构建另一个屏幕，当构建结束时，将yoffset设为400，新的屏幕将立刻显示出来。现在将开始在第一块内存区中构建下一个屏幕的数据，如此继续。这在动画中十分有用。

1.6 Framebuffer设备的信息结构(5/5)

- 另外一个应用就是用来平滑的滚动整个屏幕。就像在前面屏幕中一样，在内存分配800行的空间。每隔10毫秒设定一个定时器(timer)，将offset设为1或是更多，你将看到一个平滑滚动的屏幕。
- 将bits_per_pixel 设为1，2，4，8，16，24或32来改变颜色深度（color depth）。不是所有的视频卡和驱动都支持全部颜色深度。当颜色深度改变，驱动将自动改变fb-bitfields。它将决定在一个特定的颜色基准上，多少和哪些比特被哪种颜色使用。如果bits-per-pixel小于8，则fb-bitfields将无定义而且颜色映射将启用。
- 在fb-var-screeninfo结构结尾的定时的设置是当你选择一个新的分辨率的时候用来设定视频定时的。

1.7 操作Framebuffer的常用函数

```
int framebuffer_handler;
struct fb_fix_screeninfo fixed_info;
struct fb_var_screeninfo variable_info;
open ("/dev/fb0", O_RDWR); /*in real life, check every ioctl if it
   returns -1 */
ioctl (framebuffer_handler, FBIOGET_VSCREENINFO, &variable_info);
variable_info.bits_per_pixel = 32;
Ioctl (framebuffer_handler, FBIOPUT_VSCREENINFO, &variable_info);
ioctl (framebuffer_handler, FBIOGET_FSCREENINFO, &fixed_info);
variable_info.yoffset = 513;
ioctl (framebuffer_handler, FBIOPAN_DISPLAY, &variable_info);
```

- 这些 FBIOGET_*的ioctl命令将请求的信息写入最后一个变量所指向的结构体中。FBIOPUT_VSCREENINFO将所有提供的信息复制回内核。如果内核不能激活新的设置，将返回-1。而FBIOPAN_DISPLAY也从用户复制信息，但并不重新初始化视频模式。最好在只有xoffset或yoffset改变时使用。

1.8 操作Framebuffer的关键代码(1/2)

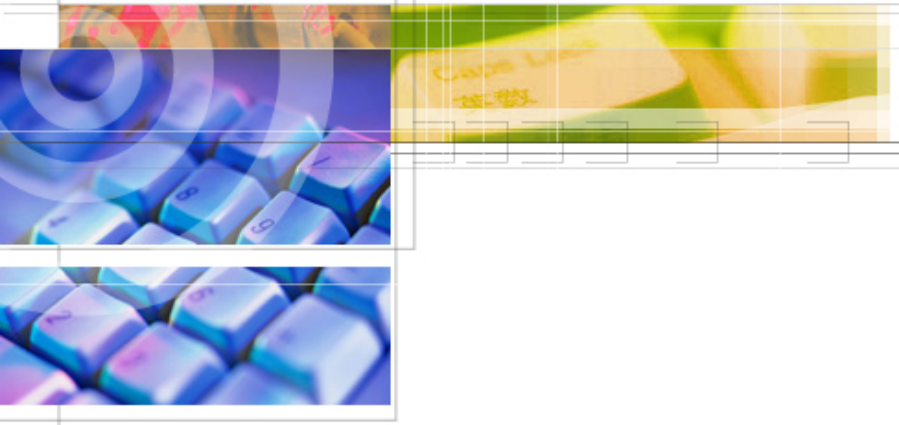
```
#include <linux/fb.h>
int main()
{
   int fbfd = 0;
   struct fb_var_screeninfo vinfo;
   struct fb_fix_screeninfo finfo;
   long int screensize = 0;
   /*打开设备文件*/
   fbfd = open("/dev/fb0", O_RDWR);
   /*取得屏幕相关参数*/
   ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo);
   ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo);
   /*计算屏幕缓冲区大小*/
   screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
   /*映射屏幕缓冲区到用户地址空间*/
   fbp=(char*)mmap(0,screensize,PROT_READ|PROT_WRITE,MAP_SHARED, fbfd,
   0);
   /*下面可通过fbp指针读写缓冲区*/
   。。。
```

1.8 操作Framebuffer的关键代码(2/2)

- 在这一部分新用到的是mmap函数。第一个变量在这种情形下可以忽略，第二个是映射的内存大小，第三个变量声明我们将共享内存进行读和写。第四个变量表示这段内存将和其他进程共享。在Framebuffer上面建一个MAP_PRIVATE是不可能的。 通常这意味着你需要中断控制台的切换去备份和恢复屏幕内容，而且不在自己没有权利的时候向屏幕内存写东西。

1.9 本次实验要求

- 在实验平台上实现对Framebuffer设备操作：
  - 在实验平台上通过直接操作Framebuffer设备实现基本绘图功能，如画线、矩形、圆等基本功能以及填充图形、显示图片等高级功能。
- 显示图片需要把图形转换成数组，这与在单片机等设备上显示图形是一样的，有一些现成的转换程序可利用；
- 显示汉字其实和显示图片是一样的，关键在于从汉字库中找到字模。



嵌入式Linux软件课程设计

华中科技大学电信学院
鄢舒
E-mail: yan0shu@gmail.com

1. 本课程考核方式

- 每人完成一项自选或指定项目的综合设计，对于复杂的项目可以多人一组，按组考核，但多人一组必须要有足够的工作量和明确分工；
- 实验结果验收与实验报告提交相结合
  - 能够全部完成的项目验收最迟在14周周一/周二（5月23日/24日）晚上上课时间完成，最终成绩将综合考虑项目的难度和完成情况；
  - 项目的源代码打包发往yan0shu@gmail.com，其中必须加上一个txt文档，内容要求有姓名、学号、班级、项目名称和必要的运行说明。

2. 实验报告的要求（1/2）

- 实验报告在课程结束后，即最迟15周星期三（6月1日）交到南一楼西320室。
- 请使用统一的实验报告首页，务必在实验报告首页加上如下内容：
  - 姓名
  - 学号
  - 班级
  - 题目

下载地址：http://pan.baidu.com/s/1i53y8DZ 密码：kixm

2. 实验报告的要求（2/2）

正文内容一般包括以下部分：

- 项目名称
- 项目需求分析：包括主要设计思路，需要完成的目标和采用的主要方法；
- 项目分工：包括同组姓名和各自分工与完成情况；
- 概要设计：包括系统整体软硬件流程图，各个功能子模块的划分和描述；
- 详细设计：仅对本人分工部分的设计进行详细描述，给出关键代码的设计思想和程序流图，并注意总结经验特别是失败的经验；
- 调试结果与改进方案：调试的方法和最终运行结果，并给出存在的问题和改进的方法；
- 参考文献：按照一般论文的方式列出。

3. 综合设计的要求

- 可以根据项目情况和自己的熟悉情况选择在嵌入式Linux操作系统下（GCC交叉编译环境）或Android系统下（Java）完成；
- 同组成员分工要明确，体现为同组实验报告不可相同，但同组源代码只需提交一份；
- 不同人的源代码完全一样的情况只认可时间上为先提交先验收的，后者等同于没有完成。

4. 综合设计的参考题目（1/3）

1. 算法类：
   - 三种或三种以上排序算法在ARM Linux上执行速度的比较：例如可以随机产生1000个数，在排序过程开始前计下系统时间，结束后再计下系统时间，算出时间差即为算法执行时间，每种算法需要多重复几次取平均值。
   - 在实验箱的LCD(Linux framebuffer设备)上完成圆或椭圆的两种以上填充算法，比较他们的填充效果和填充速度。
2. 网络类：
   - 聊天服务器程序：在实验箱上运行服务器程序，可同时接入两个以上的客户端，每个客户端有自己的标识，均可看到所有客户端的发言。

4. 综合设计的参考题目 (2/3)

- 基于web服务器的应用：在实验箱上运行web服务器，编制动态网页，利用CGI或ASP实现对系统的控制，例如实现在浏览器中编辑系统中的某个文件或执行检查网络功能(显示IP、ping等)。

3. 移植类：

- 应用程序移植：如移植MP3播放软件到实验箱，实现MP3文件的播放。
- 系统移植：如新的Bootloader或新的Linux内核及根文件系统的移植。

4. 应用类：

- 图形软件：利用控制台输入参数，在framebuffer设备上实现矩形、圆、三角形等形状绘制或图像文件的显示。

4. 综合设计的参考题目 (3/3)

- 游戏软件：如贪吃蛇、俄罗斯方块等。

5. 其他：

- 利用上课学习过的内容组合设计一个程序：如通过串口传输图像文件到LCD上显示。
- 。 。 。

5. 时间安排

- 第12周： 完成选题；
- 第13周： 完成代码编写和移植；
- 第14周周一/周二（ 5月23日/24日）晚上：完成最终验收并提交源代码文件。
- 第15周周三（6月1日）17:30前：提交纸质版实验报告。