基于 Framebuffer 的嵌入式 Linux 图形库设计①

夏　凡, 陈蜀宇, 龙昌生

(重庆大学 软件学院，重庆 400030)

摘　要：本文主要研究并设计出一套基于嵌入式体系结构的 Linux 图形库，并将其应用于项目实践中。所设计的图形库以 Linux 2.6 内核的 Framebuffer 为基础，用 C 语言编程封装构造而成，它具备矢量图形显示、BMP 位图显示、常用字符显示、窗口显示及消息响应等基本功能。

关键词：嵌入式；Linux；Framebuffer；图形库；消息循环

Designing the Embedded Linux GUI on Framebuffer

XIA Fan, CHEN Shu-Yu, LONG Chang-Sheng

(Software Engineer Department, ChongQing University, ChongQing 400043, China)

Abstract： This paper mainly researches on how to design a Graphic User Interface in embedded Linux operating system. The GUI is based on Framebuffer in Linux 2.6 kernel and is built in C programming language, its functions include displaying vector graphics, displaying bitmaps, displaying commonly used characters, displaying windows and responsing messsages. This paper shows the whole process of implementation of the GUI mentioned above.

Key words：embedded system; Linux; Framebuffer; GUI; message loop

1　引言

随着后 PC 时代的到来，嵌入式系统的性能有了大幅度的提高，应用范围也越来越广，当初的一些简单的人机交互接口已经无法满足人们的要求。而与此同时，PC 机上的图形交互界面已经普及并成熟。于是在嵌入式系统中也逐渐出现了图形用户界面，而图形用户界面的存在必然离不开系统中图形库的支持。图形库作为一种中间件软件，既不同于一般的应用软件，也不同于底层的系统软件。从层次划分上讲，它是处于这两者之间的：它通过将底层驱动程序进行抽象，加入图形化显示的功能并提供 API 接口供上层应用程序调用。

在基于 Linux 的嵌入式系统中，有不少已经比较成熟的图形库可供选择，如 MiniGUI、QT/Embedded 等。不同于裁剪移植已有的图形库，本课题将讨论如何以嵌入式 Linux 系统的 Framebuffer 机制为基础（Framebuffer 只提供对显卡的最原始操作，如将某数据写入显存的某个位置而显示某个点），将其提供的原始接口用 C 语言进行编程封装，构造一个具有图形化显示和用户交互功能的小型图形库。

2　Framebuffer原理

Framebuffer 译作帧缓冲，它作为基础图形设施，是出现在 Linux 2.2.xx 内核当中的一种驱动程序接口，是作为其他高级图形或者图形应用程序的基本函数库。这种接口将显示设备抽象为帧缓冲区。用户可以将它看成是显示内存的一个映像而不必关心物理显存的位置、换页机制等等具体细节，这些细节的实现都是由 Framebuffer 设备驱动来完成的。用户只需将 Framebuffer 映射到用户空间，就可以直接对其进行读写操作，而写操作可以立即反映在屏幕上（Framebuffer 工作原理如图 1 所示）。有了 Framebuffer，图形库设计者不需要对底层的驱动有深入了解就能够做出很好的图形。对于设计者而言，它和/dev 下面的其它设备

① 基金项目:重庆市自然科学基金(CSTC2010AB2001)
收稿时间:2011-08-25;收到修改稿时间:2011-09-13

没有什么区别。可以把 Framebuffer 看成一块内存，既可以向这块内存中写入数据，也可以从这块内存中读取数据。

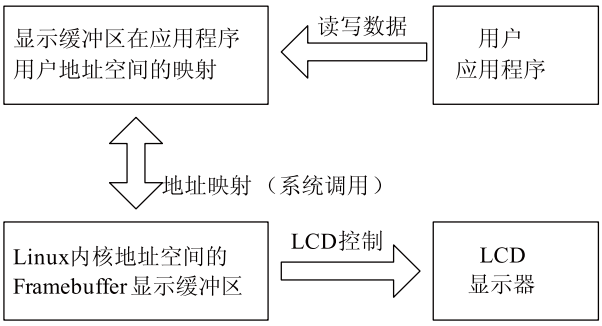

图 1 Framebuffer 工作原理

在包含 Framebuffer 驱动程序的 Linux 嵌入式系统中，对 Framebuffer 进行编程的基本步骤如下：

(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，如屏幕分辨率、色彩深度等。根据屏幕参数可计算出屏幕缓冲区的大小；

(3) 将 Framebuffer 设备文件的文件描述符映射到用户空间；

(4) 映射后就可以直接读写屏幕缓冲区，进行相应的屏幕显示；

(5) 解除屏幕映射；

(6) 关闭/dev/fb 设备文件。

综上所述，将 Framebuffer 映射到用户空间的某段内存之后，再通过操作指针往该段内存中的某个位置写入数据，就可以在屏幕上的对应位置显示对应颜色的点。Framebuffer 机制是将要构造的图形库的基础。

3 基于Framebuffer的图形库的实现

以 Framebuffer 为基础，能设计构造出一个完整的图形库。该图形库具备以下功能：

(1) 能显示基本的矢量图形；

(2) 能显示 BMP 格式的图片；

(3) 能显示常用的 ASCII 字符；

(4) 具备窗口的概念，能在窗口客户区显示图形、图片、字符；

(5) 拥有消息响应机制(主要响应触摸屏消息和按键消息)，能够响应外部输入，实现人机交互。

3.1 基本矢量图形的显示

一般而言，图形库必须具备的基本功能是显示点、线、面等矢量图形。通过 Framebuffer 的文件描述符以及映射后返回的指向屏幕第一个像素的指针，可以方便地在屏幕的指定位置画点；以点的显示为基础，运用计算机图形学的中点画线算法，即可在屏幕上画线；在实现了直线的显示之后，多边形的显示则非常容易了，只需将构成多边形的几条线段分别绘制出来即可。对于像圆这种不是由直线、而是由曲线构成的面，可以采用计算机图形学的中点画圆算法来显示。另外，本图形库还支持填充封闭的面（如三角形、矩形、圆等），这是通过种子填充算法来实现的。

3.2 BMP 位图的显示

BMP 位图文件可分为四个部分：文件头、信息头、颜色表、图像数据阵列。其中，文件头有 14 个字节，包含了 BMP 文件的标志、文件的大小以及位图数据在文件中的偏移量等信息；信息头有 40 个字节，记录了有关 BMP 位图的宽、高及压缩算法等信息；对于 16 色或 256 色的 BMP 图片文件，须使用颜色表为图像数据阵列提供颜色索引，而 24 位和 32 位的 BMP 位图文件不使用颜色表，RGB 数据直接存放在图像数据阵列里，每个像素点的数据都以 B、G、R 的顺序排列。

现在应用的 BMP 位图几乎都是 24 位的，本图形库对 24 位 BMP 位图显示的处理为：打开文件之后，先读取文件头获取该文件的类型信息，如果是 BMP 类型，再读取其信息头以获取该位图的长度和宽度信息，然后映射 Framebuffer 在内核地址空间的屏幕缓冲区到用户地址空间，最后根据 BMP 的长度和宽度复制位图文件图像数据阵列里的数据到被映射的用户地址空间，这样就会在屏幕上将该位图显示出来。

3.3 常用 ASCII 字符的显示

任何一种图形库都应该具备显示字符的能力，本图形库支持常用 ASCII 字符的显示。为了在屏幕上显示字符，系统的图形库中必须装备有相应的字符库，字符库中存储了每个字符的形状信息。字符库分为矢量型和点阵型两种：矢量型字符库采用矢量代码序列表示字符的各个笔画，输出一个字符时，系统中的字符处理器解释该字符的每个矢量代码，输出对应的矢量，达到产生字符的目的；点阵型字符库为每个字符定义一个字符掩膜，表示该字符的像素图案的一个点阵。本图形库采用点阵型字符库来实现字符的显示。

在点阵字符库中，每个字符都被定义成一个称为字符掩膜的矩阵。矩阵中的每个元素都是一位二进制位。该位为 1 时，表示字符的笔划经过此位，对应于此位的像素应被置为字符颜色；该位为 0 时，表示字符的笔划不经过此位置，对应于此位的像素应被置为背景色。

一般认为定义西文字符的掩膜矩阵尺寸应该不小于 5×7，而定义汉字字符的掩膜矩阵尺寸应该不小于 16×16。本图形库采用 16×8 的掩膜矩阵尺寸来实现常用 ASCII 字符（包括 26 个英文字母的大小写、10 个阿拉伯数字以及一些基本标点符号）的显示。

首先，需要一个数据结构来表示 16×8 的掩膜矩阵。因为 unsigned char 类型占 1 个字节，能表示 8 个数据位，所以一个长度为 16 的 unsigned char 类型的数组即可表示 16×8 的掩膜矩阵。在此之上再加上一个 char 类型（表示该掩膜矩阵对应的那个字符），并与之构成一个结构体：

```
struct character
{
    char ch;
    unsigned char font_data[16];
};
```

然后，将本图形库包含的全部字符（一共 95 个）以结构体数组的方式存储在程序的数据段之中，其中每个字符对应的掩膜矩阵的每个数据位用 0、1 标出。例如字符’!’在本图形库的字符库中存储为：

```
{
    ‘!’,
    {0x00, 0x00, 0x00, 0x10, 0x10, 0x10, 0x10,
0x10, 0x10, 0x10, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00}
}
```

根据用户提供的需要显示的字符，图形库能够遍历结构体数组以返回指向存储在数据段的与该字符对应的掩膜矩阵中的第一个 unsigned char 类型变量的指针，通过该指针能够获得字符对应的掩膜矩阵的每个数据位，然后在屏幕上的指定位置将该字符的每个数据位映射上去（如果本图形库的字符库中不包括该字符，则无法显示它，此时将其视为空字符并做相应处理）。

3.4 窗口的实现

在一般的图形库系统中，窗口是作为显示的载体而存在的。一个图形界面应用程序可以拥有一个或多个窗口，一个父窗口可以派生出一个子窗口，一个子窗口又能派生出另一个子窗口而成为它的父窗口。窗口由客户区和非客户区构成。在一个窗口中，标题栏、边框（或滚动条）就是窗口的非客户区，中间区域称为客户区。我们在窗口上进行绘制时所使用的区域就是客户区。

引入了窗口的概念，图形库系统就拥有三种坐标系统，它们之间可以转换。这三种坐标系分别是：屏幕坐标系、窗口坐标系、客户坐标系。其中屏幕坐标系以屏幕左上角为坐标原点，水平方向为 X 值，垂直方向为 Y 值；窗口坐标系以窗口左上角为坐标原点，水平方向为 X 值，垂直方向为 Y 值；客户坐标系以窗口客户区左上角为坐标原点，水平方向为 X 值，垂直方向为 Y 值。对于应用程序开发者来讲，应根据客户坐标进行图形的绘制。

由于本图形库被设计应用于嵌入式系统，而嵌入式系统中的应用程序只需要简单的图形化显示，因此仅实现单一窗口(即一个应用程序对应一个唯一的窗口)便足够。为了最大限度地节约显示空间，本图形库的窗口将不能被放大和缩小(其尺寸始终和屏幕大小一样)，而且，它仅包含窗口客户区而不包含窗口非客户区。这样一来，屏幕坐标系、窗口坐标系、客户坐标系实际上就成了相同的坐标系。

从外观上看，本图形库的窗口为一覆盖整个屏幕的白色画布，因此，在创建窗口并打开时，图形库将创建一个与屏幕尺寸相同的白色矩形来覆盖整个屏幕，同时创建一个句柄来标识该窗口。在此之后即可基于该窗口进行图形化显示。

应用程序退出时将关闭窗口，关闭成功后会销毁标识该窗口的句柄，并且返回到字符模式状态（背景色为黑色）。

3.5 消息响应机制

任何能够与用户交互的图形库系统，均有事件与消息驱动的概念。与传统的程序不同，以消息驱动的程序启动后就会一直处于循环状态，在这个循环当中，程序从外部输入设备获取某些事件，比如用户的按键或者鼠标的移动，然后根据这些事件做出某种响应，并完成一定的功能。这个循环直到程序接受到某个消息才终止。所以消息传递与消息处理是消息（事件）驱动的图形库系统的核心。消息传递与消息处理的流

程如图 2 所示：

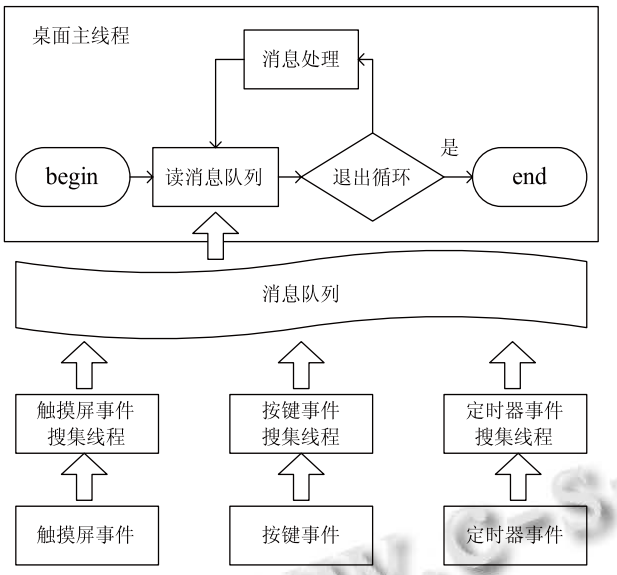

图 2 消息循环与消息处理

在消息驱动的系统中，计算机外设发生的事件都由支持系统收集。收集到的事件会被按约定的格式翻译为特定的消息。本图形库封装的消息由消息头和数据段组成：通过消息头，可以知道消息的类型（如触摸屏消息、定时器消息）；通过数据段，可以知道消息的具体内容（如触摸屏消息中被感应的屏幕坐标）。

由系统收集到的消息将被发送到图形库应用程序的消息队列中。在图形库系统中，消息是与窗口紧密联系在一起的。本图形库只提供单一的窗口，该窗口拥有一个自己的消息队列，同时有专门的线程来处理消息。消息队列是消息的暂存处，它本质上是一个先入先出的数据结构：当需要向一个窗口发送消息的时候，发送者就把消息放到目的窗口的消息队列的队尾；当目的窗口的消息队列非空时，桌面主线程将从队头取出消息进行处理。

基于本图形库的应用程序启动后，首先会初始化环境，包括创建专门用于收集外部事件的按键线程、触摸屏线程、定时器线程，然后创建桌面窗口，之后桌面主线程就会不停地循环等待消息，当收到消息后就处理这些消息。

按键线程与触摸屏线程启动后，会循环读取当前的事件，得到一个事件后，将其抽象封装成消息并发送到消息队列中。桌面主线程在收到任何一个消息后，将解析出该消息的类型，再根据应用程序的需要进行相应的处理。

消息循环实则是一死循环，图形库应用程序的主线程将不停地从消息队列中取出消息进行处理，处理完成后又去取消息（若消息队列为空则主线程会阻塞一段时间，然后再去消息队列中尝试获取新消息），直到应用程序终止的时候消息循环才随之终止。

4 结语

本文介绍了在嵌入式 Linux 系统上，如何利用 Framebuffer 机制来构造一个小型的图形库。构造的图形库能够实现基本矢量图形的显示、BMP 位图的显示、常用 ASCII 字符的显示、窗口的显示、消息响应等一系列功能。随着嵌入式软硬件系统的飞速发展，基于嵌入式的图形库系统由于其在图形化显示和人机交互上所具备的巨大优势，必将越来越多地受到人们的关注。本文介绍的这个自行开发的小型嵌入式图形库也将被逐渐完善，从而不断满足人们的需求。

参考文献

1 Neil Matthew, Richard Stones. Linux 程序设计(第 3 版). 北京:人民邮电出版社, 2007:408–501.

2 宋宝华. Linux 设备驱动开发详解.北京:人民邮电出版社, 2008:74–274.

3 孙家广.计算机图形学(第 3 版).北京:清华出版社, 1998: 165–215.

4 Greg Kroab-Hartman. Linux Kernel 技术手册. 江苏:东南大学出版社, 2007:12–15.

5 Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. 操作系统概念(第 7 版).高等教育出版社, 2007:39–78.

6 韦东山.嵌入式 Linux 应用开发完全手册.北京:人民邮电出版社, 2008:293–333.

7 郑灵翔.嵌入式Linux 系统设计.北京:北京航空航天出版社, 2008:237–241.

8 夏宝亮,张宗澄.基于 Framebuffer 的应用开发.大众科技, 2007,10:79–80.

9 Yang Jia, Ji Wankang, Hong Yongqiang. Transplantation of GTK+ Based on Framebuffer. The Eighth International Conference on Electronic Measurement and Instruments. 2007,381–384.