嵌入式 Linux 帧缓冲设备驱动程序①

赵 洁 龚 威 (天津城市建设学院 电子与信息工程系，天津 300384)

摘 要： 本文介绍了嵌入式 Linux 帧缓冲设备驱动程序的体系结构，详细分析了其核心数据结构和编写方法。结合在 ARM9 处理器 S3C2440 平台上的开发实例，讲述了 Linux2.6.33 内核基于 Platform 总线的帧缓冲设备驱动的实现原理及开发流程。最后给出了用户空间应用程序访问帧缓冲设备的一般方法。

关键词： 帧缓冲；嵌入式 Linux；LCD 驱动；Platform；ARM9

Framebuffer Driver Based on Embedded Linux

ZHAO Jie, GONG Wei (Department of Electronics and Information Engineering, Tianjin Institute of Urban Construction, Tianjin 300384,China)

Abstract: This paper introduces the systematic structure of embedded Linux framebuffer device driver, and gives a detailed analysis of the key data structures and writing method. Through the example of Linux framebuffer device driver based on ARM9-CPU S3C2440, implementation principle and development process of Linux framebuffer device driver based on Platform bus in the Linux2.6.33 kernel is analyzed. Finally, it gives the general methods about the access to framebuffer device by user-space application.

Keywords: framebuffer; embedded Linux; LCD driver; platform; ARM9

后 PC 时代的今天，嵌入式技术和多媒体应用已经深刻地改变着现代人的生活，PDA、3G 手机、车载终端等消费电子类产品大多采用 TFT LCD 屏，支持彩色图形界面和视频媒体播放。Linux 由于其开源免费、可定制、易裁剪等优势在嵌入式领域具有无可比拟的巨大市场。针对嵌入式应用的 GUI，如 QT Embedded、MiniGUI、GtkFB 等都是建立在帧缓冲(Framebuffer)设备的基础上处理与 LCD 控制器有关的底层命令。因此，嵌入式 Linux 系统中帧缓冲设备驱动程序的开发成为至关重要的一项工作。

1 Linux帧缓冲驱动程序的体系结构

帧缓冲(Framebuffer)是 Linux 系统提供的用户与 LCD 等显示设备的接口，它把显示缓冲区抽象化以此来屏蔽不同显示设备硬件的底层差异，这样上层应用程序通过直接对显示缓冲区的读写操作，即可控制 LCD 屏幕的输出[1,2]。用户可以不必关系物理显存的具体位置、换页机制等细节信息，这些都由帧缓冲设备驱动程序来完成。应用程序只要在显示缓冲区中与 LCD 显示点对应的区域写入颜色值，相应的颜色即可显示在 LCD 屏幕上。显示缓冲区的大小由屏幕分辨率和显示颜色数决定。帧缓冲设备是标准的字符设备，采用“文件层—驱动层”的接口方式，主设备号为 29，对应的设备文件为/dev/fb*。

嵌入式 Linux 帧缓冲设备驱动的开发需要填充几个重要的结构体并编写指向底层操作的成员函数，其中主要涉及内核源码中的 2 个文件：Include/Linux/Fb.h 和 Drivers/Video/Fbmem.c。

Fb.h 文件中记录了与 Linux 帧缓冲相关的重要的数据结构，并定义了 FB_MAX 宏，目前内核支持注册的帧缓冲设备数最大为 32。

1) fb_info 结构体：帧缓冲设备驱动层接口的核心数据结构，记录了关于帧缓冲设备属性、参数以及操作函数指针的完整信息。系统中每一个帧缓冲设备

① 收稿时间:2010-03-27;收到修改稿时间:2010-05-29

对应一个 fb_info 结构体，该结构体仅对内核可见。

```
struct fb_info {
    struct fb_var_screeninfo var;
/* 显示控制器的可变参数*/
    struct fb_fix_screeninfo fix;
    /* 显示控制器的固定参数*/
    struct fb_cmp cmap;      /*显示的颜色表 */
    struct fb_ops *fbops;   /*帧缓冲操作*/
……}
```

2) fb_ops 结构体：fbops 是指向帧缓冲底层操作集合的函数指针，如 fb_check_var()检查可修改的屏幕参数并调整到硬件所支持的值，fb_set_par()根据屏幕参数设置具体读写 LCD 控制器的寄存器以使其进入相应的工作状态，fb_setcolreg()设置 color 寄存器来实现伪颜色表和颜色表的填充。fb_ops 结构体的成员函数最终与 LCD 控制器硬件打交道，需要驱动程序开发人员根据 LCD 控制器的硬件设置及 LCD 显示屏的硬件参数进行编写。

3) fb_var_screeninfo 结构体：记录用户可以修改的显示控制器参数，如屏幕分辨率、每像素点的比特数、帧缓冲的 RGB 位域等信息。

4) fb_fix_screeninfo 结构体：记录固定的显示控制器的参数，如显示缓冲区的起始物理地址及长度、内存映射 I/O 的起始地址及长度等信息。

fb_info 结构体的可变参数 var 和固定参数 fix 都需要在驱动程序中进行初始化和设置。

Fbmem.c 文件中定义了帧缓冲设备的文件层接口，即提供给用户空间应用程序的 file_operations 结构体，统一实现了字符设备文件层次上的文件操作接口函数，开发时一般不需要修改；而 fb_ops 结构体的成员函数是内核空间和硬件 LCD 控制器的接口，属于驱动层，最终会操作 LCD 控制器的硬件寄存器。

```
static const struct file_operations fb_fops = {
    .owner =     THIS_MODULE,
    .read =  fb_read,   /*读函数-读屏幕缓冲区*/
    .write = fb_write, /*写函数-写屏幕缓冲区*/
    .unlocked_ioctl = fb_ioctl,/*I/O 控制函数*/
    .mmap =      fb_mmap,   /*内存映射函数*/
    .open = fb_open,   /*打开 fb 函数*/
    .release =   fb_release,   /*释放 fb 函数*/
};
```

其中 fb_mmap()函数将显示缓冲区的物理地址映射到用户空间的虚拟地址，使应用程序可以直接在图形模式下操作显示缓冲区。fb_ioctl()函数实现对用户 I/O 命令的最终执行，如获取可变屏幕参数命令 FBIOGET_VSCREENINFO、设置可变屏幕参数命令 FBIOPUT_VSCREENINFO 、设置颜色表命令 FBIOPUTCMAP 等。用户程序可以使用 FBIO****宏的 ioctl()来对帧缓冲设备进行 I/O 控制。

Linux 帧缓冲设备驱动程序的体系结构如图 1 所示，用户空间的应用程序通过 fbmem.c 文件中 file_operations 结构体统一实现的文件操作接口函数对帧缓冲设备进行访问；特定帧缓冲设备 fb_info 结构体的注册、注销以及其中 fb_ops 结构体成员函数的实现由 Linux 内核中相应的 xxxfb.c 文件实现。fb_ops 结构体的成员函数作为驱动层的核心模块实现对应的帧缓冲操作，最终直接操作 LCD 控制器的硬件寄存器。

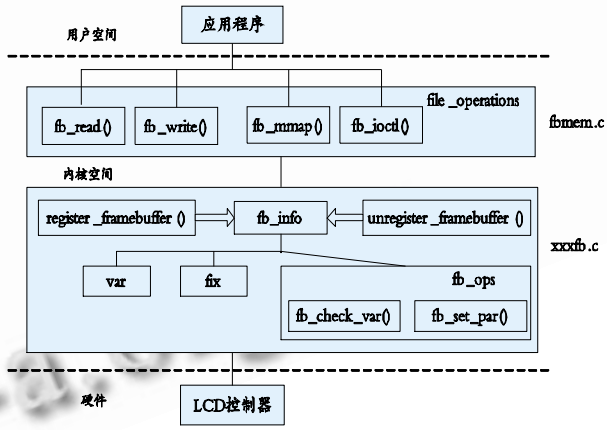

图 1 Linux2.6 内核帧缓冲设备驱动程序的体系结构[3]

2 Linux2.6内核中帧缓冲设备驱动的实现

Linux2.6 内核开始引入了基于 Platform 的新的驱动管理和注册机制。Platform 是一种虚拟总线，在嵌入式系统中用于连接 SoC 集成的片上资源到 CPU 总线上，具有更好的可移植性[4]，因此 Linux2.6 内核中的大部分驱动都依照 Platform 机制改写，设备用 platform_device 表示，驱动程序用 platform_driver 进行注册。相比传统的 driver_register()注册机制，Platform 机制的优势在于将设备本身的资源注册进内核，由内核统一进行管理，在驱动程序中使用这些

资源时可以通过 platform device 提供的标准接口进行申请并使用，这样将驱动和资源管理分离，提供了驱动程序的可移植性。

基于 Platform 总线的驱动开发步骤为：首先定义并注册 platform_device；然后定义并注册 platform_driver。Platform driver 遵循设备模型的约定，它的发现和计数都是由系统的驱动注册机制完成，驱动开发人员只需要初始化必须的数据结构并调用注册驱动的内核 API 就可以了。

下面以 ARM9 处理器 S3C2440 中 LCD 控制器为例，介绍 Linux2.6.33 内核中基于 Platform 总线的帧缓冲设备驱动的实现原理及开发流程。作为 S3C2440 片上的独立硬件模块，LCD 控制器即为平台设备，其主要功能是传输显示数据并产生控制信号，驱动程序只需要通过读写一系列寄存器即可完成配置和显示控制。

1) 定义平台设备 platform_device。Linux2.6.33 内核 Arch/Arm/Plat-s3c24xx/Devs.c 文件定义了 s3C24xx 平台设备大部分板级资源，是一个高度可移植的文件。其中定义了 LCD 控制器 struct platform_device s3c_device_lcd。

2) 注册平台设备 platform_device。Linux2.6.33 内核 Arch/Arm/Mach-s3c2440/Mach-smdk2440.c 文件中定义了 smdk2440_devices 将系统资源组织起来，通过调用 platform_add_devices()函数统一注册进内核。

3) 定义平台驱动 platform_driver，用来驱动该设备。Drivers/Video/S3c2410fb.c 文件中定义了

```
static struct platform_driver s3c2410fb_driver = {
    .probe = s3c2410fb_probe,  /*平台驱动探测函数*/
    .remove = s3c2410fb_remove,/*平台驱动移除函数*/
    .suspend     = s3c2410fb_suspend,
                                /*平台驱动挂起函数*/
    .resume = s3c2410fb_resume,
                                /*平台驱动恢复函数*/
     .driver  = {
         .name   = "s3c2410-lcd",
                    /*平台驱动名称*/
         .owner = THIS_MODULE,  },
};
```

4) 注册平台驱动 platform_driver。Linux 2.6.33 内核 Drivers/Video/S3c2410fb.c 文件中通过函数 int __init s3c2410fb_init(void) 调用 platform_driver_register(&s3c2410fb_driver) 完成平台驱动的注册。通过 module_init (s3c2410fb_init)宏进行驱动程序模块的初始化。

驱动模块加载函数中对平台驱动的注册会引发其中探测函数 s3c2410fb_probe()的执行，它主要完成 4 项工作：

1) 申请 struct fb_info 内存空间并初始化 struct fb_info 的 fb_var_screeninfo 和 fb_fix_screeninfo 结构体；

2) 根据不同 LCD 屏幕参数，完成 LCD 控制器硬件的初始化；

3) 申请显示缓冲区的内存空间；

4) 注册帧缓冲设备；

相反，在驱动模块卸载函数中对平台驱动的注销会引发其中移除函数 s3c2410fb_remove()的执行，它释放显示缓冲区并注销帧缓冲设备。

3 应用程序访问帧缓冲设备的方法

帧缓冲设备是一种典型的字符设备，应用程序可以通过/dev 目录下的设备文件节点对其进行操作。应用程序通过 mmap()映射操作将显示缓冲区的物理地址映射到用户空间的一段虚拟地址后，通过读写这段虚拟地址来实现对显示缓冲区的访问，这样就可以在屏幕上进行绘图。同时，应用程序可以通过 ioctl()操作读取或设置显示设备及屏幕的参数，如屏幕分辨率、屏幕大小、每像素点比特数和偏移以及颜色数等。

一般来说，用户空间的应用程序操作帧缓冲设备可以分 3 个步骤：

1) 打开设备文件/dev/fb*并利用 ioctl()操作获取屏幕可变参数和固定参数。

2) 根据获得的屏幕分辨率和每个像素的比特数(bpp 值)计算出屏幕缓冲区的大小，并利用 mmap()操作将屏幕缓冲区映射到用户空间。

3) 在用户空间直接读/写显示缓冲区完成绘图等

操作。

应用程序访问帧缓冲设备的关键代码框架如下,其中省略了系统调用返回值的错误检验[5,6]:

```
……
struct fb_var_screeninfo var;
int fd = -1,bp = 0;
char *fbbuf = NULL;
fd = open(getenv("FRAMEBUFFER"),O_WRONLY);
                                /*打开帧缓冲设备文件*/
ioctl(fd,FBIOGET_VSCREENINFO,&var);
                                /*获取屏幕可变参数*/
……                              /*校正 panning 及 offset*/
fbbuf = convertRGB2FB
(fd,rgbbuf,x_size                                *
y_size,var.bits_per_pixel,&bp);
                /*将 RGB 颜色数据按照屏幕的 bpp 值调整 */
blit2FB(fd,fbbuf,x_size,y_size,var.xres,var.yres,
x_pan,y_pan,x_offs,y_offs,bp)
                                /*写显示缓冲区*/
free(fbbuf);                    /*释放显示缓冲区*/
close(fb);                      /*关闭帧缓冲设备文件*/
```

4 结语

帧缓冲设备驱动的开发是当今嵌入式 Linux 系统产品开发的重要工作之一，深入理解 Linux2.6 内核基于 Platform 机制的帧缓冲驱动的开发流程有助于在实际的项目研发中，根据不同显示设备的硬件手册编写帧缓冲设备驱动程序。由于帧缓冲设备文件层的接口函数由 Linux 内核 Fbmem.c 文件中的 file_operations 结构体统一实现，从而驱动开发人员的工作重点是实现针对特定显示设备的帧缓冲操作即 fb_info 中 fb_ops 结构体的成员函数，其中 fb_info 中的可变参数 var 非常关键，它和 LCD 控制器的硬件设置以及 LCD 显示屏的硬件参数相对应。本文基于 Linux2.6.33 内核深入探讨了嵌入式 Linux 帧缓冲设备驱动程序的体系结构及实现原理，总结了用户空间应用程序访问帧缓冲设备的一般方法，对于实际项目中帧缓冲设备驱动的开发具有较好的参考价值。

参考文献

1 Corbet J, Rubini A, Kroah G -Hartman. Linux Device Drivers，Third Edition. USA.O Reilly Media，Inc. 2005.

2 刘淼.嵌入式系统接口设计与 Linux 驱动程序开发.北京:北京航空航天大学出版社, 2006.

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

4 博韦,西斯特. 深入理解 Linux 内核(第三版).北京:中国电力出版社, 2007.

5 Neil Matthew, Richard stones. Beginning Linux Programming 4th Edition. Wiley Publishing, Inc. 2007.

6 李云华.独辟蹊径品内核：Linux 内核源代码导读.北京:电子工业出版社, 2009.