華中科技大學

嵌入式实验报告

学 生 姓 名: 刘本嵩

学 号: U201614531

专 业: 计算机科学与技术

班 级: CS1601

指 导 教 师: 张杰

2019 年 6 月 20 日

目录

实验 1　核心编译，系统烧录

1.1　实验内容

1. 系统镜像的编译生成
   - Uboot 编译（不要求）
   - Kernel 编译
   - Android 系统编译（不要求）
2. Android+Linux 系统烧录
   - 串口工具 cutecom 使用，控制 uboot
   - usb 烧写工具 fastboot 使用
3. 简单 Linux 应用程序开发
   - 使用 Android NDK 编译简单应用程序
   - 使用 usb 开发工具 adb 上传并运行程序

1.2　实验设计

首先熟悉 Linux Kernel 目录结构

- arch：与体系结构有关的代码，zimage 在 arch/arm/boot 下；
- drivers：包括所有的驱动程序
- fs：各种文件系统格式支持源码；
- ipc：System V 的进程通信实现，包括信号量，共享内存；
- kernel：进程调度，创建，定时器，信号处理等；
- mm：内存管理；
- net：套接字和网络协议的实现；

然后使用 make menucongif 命令，图像化配置内核。接着在核心源代码根目录下执行 make zImage 命令生成内核镜像 zImage。然后开始系统烧写的准备工作。在用户根目录下编辑 adb_usb.ini 文件。若没有该文件，创建一个。接着进行设备的连接工作。

1.3 实验过程

首先修改 common/rules.mk，增加便于使用的规则 burn，同时修改使用更好的编译选项。下面仅列出修改部分，且在下文不再重复。

```
CFLAGS:=--sysroot=$(DIR)/platforms/android-9/arch-arm -march=armv7-a
    -mfloat-abi=softfp -mfpu=neon -O1 -Wall -Wextra -std=c99
burn: clean $(EXENAME)
  adb push $(EXENAME) /data/local/$(EXENAME) && adb shell /data/local/$(EXENAME)
```

1. 为 ARM 编译安装 Linux 内核。
    - make xconfig # 我喜欢 xconfig
    - make zImage
    - 连接好设备，打开 cutecom，按要求进入 fastboot 模式，下面开始写系统。
    - sudo fish
    - fastboot flash kernel zImage # 已经在 root，且系统已经安装 fastboot。无需从目录启动旧版。
    - fastboot flash ramdisk ramdisk-uboot.img
    - fastboot -w
    - fastboot flash system system.img
    - 下面使用 cutecom 输入 bootargs 指令。
    - setenv bootcmd movi read kernel 0 40008000;movi read rootfs 0 41000000 100000;bootm 40008000 41000000
    - setenv bootargs
    - saveenv
2. 编译生成 lab1：
    - make
3. 把文件上传到实验板上的/data 目录并运行：
    - make burn

实验 2　Linux framebuffer 界面显示开发

2.1　实验内容

1. Linux 下的 LCD 显示驱动接口：framebuffer 的使用原理。
2. 基本图形的显示：点、线、矩阵区域。
3. 双缓冲机制

2.2　实验设计

首先了解 Linux framebuffer 驱动原理。

- 通过 framebuffer，应用程序用 mmap 把显存映射到程序虚拟地址空间，将要显示的数据写入写个内存空间就可以在屏幕上显示出来。
- 驱动程序分配系统内存作为显存；实现 file_operation 结构中的接口，为应用程序服务；实现 fb_ops 结构中的接口，控制和操作 LCD 控制器。
- 驱动程序将显存的起始地址和长度传给 LCD 控制器的寄存器（一般由 fb_set_var 完成），LCD 控制器会自动的将显存中的数据显示在 LCD 屏幕上。

实验中需要使用的双缓冲机制如下：

一般情况下，最终的用户界面都需要经过若干次绘图才能完成，比如要先擦除之前的界面内容，再绘制新的界面内容，如果这些中间绘图是直接在 framebuffer 上操作，那么在 LCD 屏幕上就会看到这些中间结果。比如会看到屏幕先被清除，再显示出来界面，而不是界面内容直接出现在屏幕上。这就是屏幕闪烁。解决屏幕闪烁的办法就是双缓冲，所有的绘图都先绘制在一个后缓冲中（后缓冲：和 framebuffer 同样大小的一块内存）。绘制完毕后再把最终屏幕内容拷贝到 framebuffer 中。

双缓冲的绘图过程：

- 所有的绘图函数都在后缓冲中绘图。
- 所有的绘图函数都要记录本次的绘图区域：void _update_area(int x, int y, int w, int h)
- 绘图完毕后，把后缓冲中所有需要更新的绘图区域内容拷贝到前缓冲: void fb_update(void);
- 清空需要更新的绘图区域;

双缓冲机制的扩展:

- 前后缓冲可以交换: 前缓冲内存内容对应屏幕的显示，前缓冲内存的首地址是可以修改的 (显卡驱动支持，一个寄存器的内容) 后缓冲绘制完之后，交换前后缓冲
- 帧同步信号（垂直同步信号 VSYNC)：硬件 DMA 周期性 (60Hz) 的扫描读取前缓冲的内存，传递给 LCD 控制器显示。每次完整传输完一帧图像都有一个帧同步信号，然后等待大约 16ms 之后再重新开始。

2.3 实验过程

2.3.1 通用优化

首先，每个画点函数都调用 update 将会浪费大量 CPU 时间。我实现了一个 raw_draw_pixel，在其他绘画函数中调用它，而不是重新检查边界。这将大大加快运行速度。

其次，main 中过快的刷新率使得 CPU 为人眼根本无法看清的细节浪费了大量 CPU 时间。我调整了 fb_update，减慢了不必要的刷新，使得画点、画线、画矩形都能在人眼无法分辨的情形下将时间缩短到 2 毫秒内。(被要求移除此优化)

其次，上一节中在编译选项中加入-O3 和-std=c99，不但速度加快，还发现了库中的若干软件错误。下一次实验将会详细描述这些错误。

2.3.2 画线函数

画线函数的大致思路为：首先计算所划线的斜率的绝对值，若斜率的绝对值大于一，则交换 x1 和 y1 的值，x2 和 y2 的值。画线的时候对于所画线的斜率分为两种情况，斜率绝对值大于 1 的情况和斜率绝对值小于一的情况。具体的实现就是按照计算得出的斜率值，每次画出一个点以后，就使用斜率值计算出下一个点的位置，然后调用画点函数进行画点，直到画出所有的点。

2.3.3 画矩形函数

矩形函数的主要思路为：首先检查输入的坐标是否越界，然后调用 update 函数，使用两个 for 循环绘制所有点即可。

实验 3　图片显示和文本显示

3.1　实验内容

1. JPEG 不透明图片显示
2. PNG 半透明图片显示
3. 矢量字体显示：
   (a) 字模的提取
   (b) 字模的显示（只有 alpha 值的位图）

3.2　实验设计

同实验二相似，实验三也需要我们直接操作 FrameBuffer 将屏幕上每一个像素点的 RGB 值更新。

实验框架已经提供了相应的图片、字体解析接口将图片和字体转换成统一的、可以被直接解析的格式；实验框架也提供了基于现实字体封装的显示字符串的函数。

在实验三中，我们无需关注图片、字体格式的细节（图片编码、字模），只需处理调用相应接口后返回的`fb_image`即可。

需要注意的是，PNG 图片和字体需要能支持 Alpha 通道（即透明度通道），透明度叠加的公式实验讲义已经给出。

3.3　实验过程

3.3.1　JPEG 图片显示

JPEG 的图片无需计算透明度，而且其格式与 FrameBuffer 的格式相似，可以直接采用单循环按行进行内存拷贝。该实现缓存友好、效率较高。显示 JPEG 图片的流程如图**??**所示。

3.3.2　PNG 图片显示

PNG 图片由于涉及透明度，所以显示时采用的方式是依次遍历待显示每一个像素点，根据像素点的 Alpha 值以及对应 FrameBuffer 的三通道（R、G、B）值分别新的通道值并写入 FrameBuffer。

当 Alpha 值为 0 或是 255 时，无需考虑 Alpha 值以及相关的计算，以节省计算资源。

实际上，可以把每一个 M*N 图片转换成三个 M*N 的颜色矩阵——R 矩阵、G 矩阵、B 矩阵。所有的透明度计算均可以变为矩阵的相关运算。这样可以充分利用缓存的特性、同时优化运算。出于时间和难度等原因，未采用这种实现。

有其他同学提到，在实现时遇到了程序 SegmentFault 的错误，并且确定了原因是：

1. 未对边界进行合理的检查，导致出现了超过 FrameBuffer 的地址的写入。
2. 程序开启了-O3 编译优化参数，部分激进的优化使得程序出错。

其实这种调查结论只是权宜之计。我也开启了-O3，同时开启了 c99 标准发现了以下致命错误：

1. common/external.c 没有 include time.h
2. common/external.c:fb_new_image 函数有两处本该返回 NULL 或 image，却漏写返回值。C89 以神奇的容错性允许了这种写法。
3. common/external.c:fb_get_sub_image 函数的 i 应当是 unsigned int。
4. 其他非致命问题。

只需修复这些错误，程序的行为自然恢复可控。实验中明明使用了虽然落后但支持 c99 的 gcc4 编译器，大家却都在使用老旧的 c89 标准，它甚至不会发出 warning，以至于无人发现第二条的致命错误。

3.3.3 字体显示

字体显示同 PNG 显示几乎一样。但是在实现时遇到了字体宽度只有原字体宽度 1/4 的情况。这是因为字体图片和 PNG 图片的格式不同。字体图片中的每一个字节只代表 Alpha 值，而 R、G、B 值由函数入口参数 color 提供。因此还需要对字体显示的部分代码进行修改，补齐缺少的 R、G、B 三通道颜色的计算。修改完成后，实验结果正确。

实验 4　Linux touchscreen 多点触摸开发

4.1　实验内容

1. Linux 下的触摸屏驱动接口:
   (a) Input event 的使用
   (b) 多点触摸协议 (Multi-touch Protocol)
2. 获取多点触摸的坐标
3. 在 LCD 上显示多点触摸轨迹
4. 绘制一个清除屏幕的按钮，点击后清除屏幕内容

4.2　实验设计

Linux 下的触摸屏驱动接口以及实验相关的框架代码已经给出。设备驱动的初始化也已经提供。其本质是一个忙等待的无限循环，在循环内部，程序主动地调用相关函数，获取当前的触摸事件。

触摸事件分为三类——TOUCH_PRESS、TOUCH_MOVE、TOUCH_RELEASE，分别代表手指触碰，手指移动和手指离开。最多可以同时识别 5 个不同的触摸点。实验选做第一部分——即显示一个背景图片，在背景图片上根据手指的轨迹画线，要求:

1. 轨迹是连贯的，不能断断续续
2. 绘制一个清除屏幕的按钮，点击后清除屏幕内容

根据实验的要求，将整个程序划分为主模块、清空模块、以及绘制模块。

- 主模块负责通过触摸板驱动接口轮询以获取触摸事件并调用其他模块。
- 清空模块提供一个清空函数，重绘整个屏幕（包括背景图片和清空按钮）。
- 绘制模块负责在两点间绘制连贯的线。

4.3 实验过程

4.3.1 清空

我的晴空功能只需要通过 draw_button 函数即可实现。它画出一个巨大的矩形清空屏幕并放置一个按钮。

4.3.2 绘制

我将 touch_read 分为两个部分。touch_read1 是原 touch_read 函数，on_touch_read_finish 负责连点成线。其使用数组保存多个手指的历史位置。如果发现点击的位置在按钮内，则触发清空函数。否则，它将根据触摸点编号，在当前点和上一次的该触摸点之间画一条线。

4.3.3 测试

在测试时，出现了画出的线断断续续的问题，经过长时间排查，最终确定了原因是画线函数的问题。

通过临时修改画线函数后，程序行为恢复正常。

实验 5　Linux LED 驱动和控制界面

5.1　实验内容

1. 编写 LED 驱动: 初始化、LED 控制函数;
2. 使用模块方式编译、安装驱动;
3. 编写测试程序, 绘制界面并控制 LED 驱动;

5.2　实验设计

由于 LED 的闪烁不能受到画面绘制、手势识别等界面有关的程序的影响，因此需要 2 个线程或进程共同完成这一任务。为了便于通信以及程序的编写，最终决定采用两个县城完成这一任务。线程间通信的内容为 LED 灯的闪烁频率或闪烁时间间隔。两个线程之间的关系如图5.1所示。从图中可以清晰的看到，对于通信内容频率值而言，由于生产者（主线程）只有一个，消费者（控制 LED 闪烁的线程）也只有一个，因此在这种情况下进行线程间通信不需要加锁。幸运的是，这个 linux 上的 pthread 库可以正常使用。我的多线程由 pthread 实现。

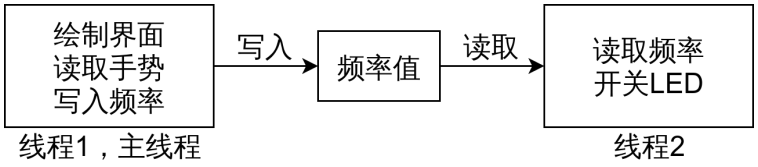

图 5.1: 线程之间的关系

在此实验中，除了背景外其余的控件均是使用函数直接进行绘制。一共包含 5 个控件：滑槽、滑槽填充、滑块、弹出框、百分比，其形状以及图层如图**??**所示，图层越高的控件越后绘制。

于是，整个程序被分为了 2 个模块：界面设计模块以及 LED 控制模块。对于界面模块而言，其需要运行一个主循环，在循环内不断的判断手势以及手势的位置，然后根据触点的位置判断按钮是否被按下、滑块是否被按下等等。两个线程的流程图如图**??**所示。

5.3 实验过程

在程序编写完成后，首先使用 make 编译，然后通过 make burn 上传开发板并运行。初次运行时发现无论如何拖动滑块 LED 灯一直为常亮状态。经过仔细的分析源代码后发现这是由于 LED 闪烁间隔设置得太小造成的，LED 闪烁的频率过快导致人眼看起来像是常亮状态。在修改闪烁间隔后，LED 能够在人眼能观察到的范围内变化闪烁频率。并在滑块被拖动到两个端点时能够停止闪烁或到达常亮状态。

实验心得

通过这次实验，我对于嵌入式系统有了更为深入的了解，从最开始的系统烧录到最后的上层应用，我对于架构有了更为深入和完整的认知。使我了解了不同的嵌入式平台的不同架构特性，也体会到了 linux 系统对于在嵌入式平台的优劣。而通过后面对于图形界面的构建，不仅使我了解了平时在其他实验中通过调库构建的图形界面在底层是如何实现的、event loop 的具体实现，也使我对于 linux 底层架构，对于“一切皆文件”这一概念有了更为深入的了解。

这次难度逐级加大的实验对于我最深刻的影响就是对于包括嵌入式 linux 在内的 linux 设备文件的了解。实验 1 ~ 3 的内容都能被轻松的移植到 x86 linux 平台运行，也使我感慨将设备抽象成文件的便利性。对于图形界面的绘制也令我体会到了各种绘图算法的原理，并能够在将来的学习生活中加以应用。

实验代码

common/graphic.c

```c
#include "common.h"
#include <linux/fb.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <string.h>

static int LCD_MEM_BUFFER[SCREEN_WIDTH * SCREEN_HEIGHT];
static int *LCD_FRAME_BUFFER = NULL;

#define PURPLE FB_COLOR(139,0,255)
#define RED  FB_COLOR(255,0,0)
#define FB_COLOR(r,g,b) (0xff000000|(r<<16)|(g<<8)|b)
```

```
16
17
18  static struct {
19      int x1, y1, x2, y2;
20  } update_area = {0,0,0,0};
21
22  void fb_init(char *dev)
23  {
24      int fd;
25      struct fb_fix_screeninfo fb_fix;
26      struct fb_var_screeninfo fb_var;
27
28      if(LCD_FRAME_BUFFER =/= NULL) return; /*already done*/
29
30      //First: Open the device
31      if((fd = open(dev, O_RDWR)) < 0){
32          printf("Unable to open framebuffer %s, errno = %d\n", dev, errno);
33          return;
34      }
35      if(ioctl(fd, FBIOGET_FSCREENINFO, &fb_fix) < 0){
36          printf("Unable to FBIOGET_FSCREENINFO %s\n", dev);
37          return;
38      }
39      if(ioctl(fd, FBIOGET_VSCREENINFO, &fb_var) < 0){
40          printf("Unable to FBIOGET_VSCREENINFO %s\n", dev);
41          return;
42      }
43
44      printf("framebuffer info: bits_per_pixel=%u width=%u height=%u
            line_length=%u smem_len=%u\n",
45          fb_var.bits_per_pixel, fb_var.xres, fb_var.yres, fb_fix.line_length,
                fb_fix.smem_len);
46
47      //Second: mmap
48      int *addr;
49      size_t size = fb_var.xres * fb_var.yres * fb_var.bits_per_pixel/8;
50      addr = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
51      if((int)addr == -1){
52          printf("failed to mmap memory for framebuffer.\n");
53          return;
54      }
55      LCD_FRAME_BUFFER = addr;
56      return;
57  }
58
59
```

```
60 static void _update_area(int x, int y, int w, int h)
61 {
62    //if((w <= 0)||(h <= 0)) return; /* sure */
63    int x2 = x+w;
64    int y2 = y+h;
65    if(update_area.x2 == 0) {
66       update_area.x1 = x;
67       update_area.y1 = y;
68       update_area.x2 = x2;
69       update_area.y2 = y2;
70    } else {
71       if(update_area.x1 > x) update_area.x1 = x;
72       if(update_area.y1 > y) update_area.y1 = y;
73       if(update_area.x2 < x2) update_area.x2 = x2;
74       if(update_area.y2 < y2) update_area.y2 = y2;
75    }
76    return;
77 }
78
79 /** copy data from mem buffer to frame buffer */
80 void fb_update(void)
81 {
82    if(LCD_FRAME_BUFFER == NULL){
83       printf("error: not allocate space for frame buffer\n");
84       return;
85    }
86
87    if((update_area.x1 >= SCREEN_WIDTH)
88       || (update_area.x2 <= 0)
89       || (update_area.y1 >= SCREEN_HEIGHT)
90       || (update_area.y2 <= 0)) return;
91
92    int x,y,w,h;
93    x = (update_area.x1 < 0) ? 0 : update_area.x1;
94    y = (update_area.y1 < 0) ? 0 : update_area.y1;
95    w = (update_area.x2 > SCREEN_WIDTH)?
96       SCREEN_WIDTH - x : update_area.x2 - x;
97    h = (update_area.y2 > SCREEN_HEIGHT)?
98       SCREEN_HEIGHT - y : update_area.y2 - y;
99
100   int *src, *dst;
101   src = LCD_MEM_BUFFER + y*SCREEN_WIDTH + x;
102   dst = LCD_FRAME_BUFFER + y*SCREEN_WIDTH + x;
103   while(h-- > 0){
104      memcpy(dst, src, w*4);
105      src += SCREEN_WIDTH;
```

```
106         dst += SCREEN_WIDTH;
107     }
108 
109     update_area.x2 = 0;
110     return;
111 }
112 
113 
114 // static inline void _update_area(int x, int y, int w, int h) {}
115 
116 /*====================================================================*/
117 
118 void fb_draw_pixel(int x, int y, int color)
119 {
120     if(x<0 || y<0 || x>=SCREEN_WIDTH || y>=SCREEN_HEIGHT) return;
121     _update_area(x,y,1,1);
122     int *tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
123     *tmp = color;
124     return;
125 }
126 static __attribute__((hot)) __attribute__((always_inline)) inline void
        fb_raw_draw_pixel(int x, int y, int color) {
127     int *tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
128     *tmp = color;
129 }
130 
131 void fb_draw_rect(int x, int y, int w, int h, int color)
132 {
133     if(x < 0) { w += x; x = 0;}
134     if(x+w > SCREEN_WIDTH) { w = SCREEN_WIDTH-x;}
135     if(y < 0) { h += y; y = 0;}
136     if(y+h >SCREEN_HEIGHT) { h = SCREEN_HEIGHT-y;}
137     if(w<=0 || h<=0) return;
138     _update_area(x,y,w,h);
139 /*----------------------------------------------------*/
140     for(int y_shift = 0; y_shift < h; ++y_shift) {
141         // fuck a line
142 #ifdef wmemset
143         if(sizeof(wchar_t) == sizeof(int))
144             wmemset(LCD_MEM_BUFFER + SCREEN_WIDTH * (y+y_shift), (wchar_t)color,
                    w);
145         else
146 #endif
147         {
148             for(int x_shift = 0; x_shift < w; ++x_shift) {
149                 fb_raw_draw_pixel(x + x_shift, y + y_shift, color);
```

```
150             }
151         }
152     }
153 /*-------------------------------------------------*/
154     return;
155 }
156
157 void fb_clear_screen() {
158     _update_area(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
159     for(int *ptr = LCD_MEM_BUFFER; ptr < LCD_MEM_BUFFER + SCREEN_WIDTH *
            SCREEN_HEIGHT; ++ptr) {
160         *ptr = 0;
161     }
162 }
163
164 #define max(a, b) ((a > b) ? (a) : (b))
165 #define min(a, b) ((a > b) ? (b) : (a))
166
167 void fb_draw_line(int x1, int y1, int x2, int y2, int color)
168 {
169     int min = (x1<x2)?x1:x2;
170     int max = (x1<x2)?x2:x1;
171     int x = min;
172     int ymin=(x1<x2)?y1:y2;
173     int ymax=(x1<x2)?y2:y1;
174     int y =ymin;
175     if(x1==x2){
176         if(x >= SCREEN_WIDTH) return;
177         for(; y1<=y2; y1++){
178             if(y1 >= SCREEN_HEIGHT) break;
179             fb_raw_draw_pixel(x, y1, color);
180         }
181         return;
182     }
183     double k = (y2*1.0-y1*1.0)/(x2*1.0-x1*1.0);
184
185     if(k>=2 || k<=-2){
186         double x0 = x;
187         for(; x0<=max; x0+=0.1)
188         {
189             fb_raw_draw_pixel(x0, k*(double)(x0-min)+(double)y, color);
190
191         }
192
193     }
194
```

```
195     else{
196         for(; x<=max; ++x)
197         {
198             fb_raw_draw_pixel(x, k*(double)(x-min)+(double)y, color);
199         }
200     }
201 
202 
203     /*画线函数*/
204     return;
205 }
206 
207 //void fb_draw_line(int x1, int y1, int x2, int y2, int color)
208 //{
209 // const int w = abs(x1 - x2);
210 // const int h = abs(y1 - y2);
211 // const int xbegin = min(x1, x2), ybegin = min(y1, y2);
212 // _update_area(xbegin, ybegin, w, h);
213 ///*-------------------------------------------------*/
214 // if(w > h) {
215 //     const float y_per_x = (float)(y2 - y1) / (x2 - x1);
216 //     const int xend = max(x1, x2);
217 //     const int y_for_xbegin = xbegin == x1 ? y1 : y2;
218 //     for(int curr_x = xbegin; curr_x <= xend; ++curr_x) {
219 //         const int curr_y = y_for_xbegin + (int)(y_per_x * (curr_x - xbegin));
220 //         fb_raw_draw_pixel(curr_x, curr_y, color);
221 //         fb_raw_draw_pixel(curr_x, curr_y+1==SCREEN_HEIGHT?curr_y-1:curr_y+1,
    color);
222 //     }
223 // }
224 // else {
225 //     const float x_per_y = (float)(x2 - x1) / (y2 - y1);
226 //     const int yend = max(y1, y2);
227 //     const int x_for_ybegin = ybegin == y1 ? x1 : x2;
228 //     for(int curr_y = ybegin; curr_y <= yend; ++curr_y) {
229 //         const int curr_x = x_for_ybegin + (int)(x_per_y * (curr_y - ybegin));
230 //         fb_raw_draw_pixel(curr_x, curr_y, color);
231 //         fb_raw_draw_pixel(curr_x+1==SCREEN_WIDTH ? curr_x-1 : curr_x+1,
    curr_y, color);
232 //     }
233 // }
234 ///*-------------------------------------------------*/
235 // return;
236 //}
237 
238 
```

```
239 void mycircle(int x, int y, int r, int color)
240 {
241    int ix = x - r;
242    int iy = y - r;
243    int size = r + r;
244    int dx = 0;
245    int dy = 0;
246 
247    _update_area(ix,iy,size,size);
248 
249    int i,j;
250    for(i = 0;i < size;i++){
251       for(j = 0;j < size;j++){
252          dx = ix - x;
253          dy = iy - y;
254         if(dx < 0){
255         int *tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * (iy + i) + ix + j;
256          *tmp = color;
257         }
258 
259       }
260    }
261    return;
262 }
263 
264 
265 
266 
267 /*====================================================================*/
268 
269 void fb_draw_image(int x, int y, fb_image *image, int color)
270 {
271    if(image == NULL) return;
272 
273    int ix = 0; //image x
274    int iy = 0; //image y
275    int w = image->pixel_w; //draw width
276    int h = image->pixel_h; //draw height
277 
278    if(x<0) {w+=x; ix-=x; x=0;}
279    if(y<0) {h+=y; iy-=y; y=0;}
280 
281    if(x+w > SCREEN_WIDTH) {
282       w = SCREEN_WIDTH - x;
283    }
284    if(y+h > SCREEN_HEIGHT) {
```

```
285         h = SCREEN_HEIGHT - y;
286     }
287     if((w <= 0)||(h <= 0)) return;
288 
289     _update_area(x,y,w,h);
290 
291     char *dst = (char *)(LCD_MEM_BUFFER + y*SCREEN_WIDTH + x);
292     char *src = image->content + iy*image->line_byte + ix*4;
293 /*-----------------------------------------------------------------*/
294 
295     int alpha;
296     int ww;
297 
298     if(image->color_type == FB_COLOR_RGB_8880) /*lab3: jpg*/
299     {
300     // printf("you need implement fb_draw_image() FB_COLOR_RGB_8880\n");
          exit(0);
301 
302         for(iy=0;iy<h;iy++){
303 
304             for(ix=0;ix<w;ix++){
305             dst = (char *)(LCD_MEM_BUFFER + iy*SCREEN_WIDTH + ix);
306             src = image->content + iy*image->line_byte + ix*4;
307             switch(src[3]){
308             case 0:break;
309             case 255:{
310             dst[0]=src[0];
311             dst[1]=src[1];
312             dst[2]=src[2];
313             }
314             default:{
315             dst[0]+=(((src[0]-dst[0])*src[3])>>8);
316             dst[1]+=(((src[1]-dst[1])*src[3])>>8);
317             dst[2]+=(((src[2]-dst[2])*src[3])>>8);}
318             }
319 
320             }}
321         return;
322     }
323 
324     if(image->color_type == FB_COLOR_RGBA_8888) /*lab3: png*/
325     {
326         //printf("you need implement fb_draw_image() FB_COLOR_RGBA_8888\n");
              exit(0);
327 
328         for(iy=0;iy<h;iy++)
```

```
329             {
330         for(ix=0;ix<w;ix++)
331             {
332                 dst = (char *)(LCD_MEM_BUFFER + (iy+y)*SCREEN_WIDTH + ix+x);
333                 src = image->content + (iy)*image->line_byte + ix*4;
334 
335         switch(src[3]){
336         case 0:break;
337         case 255:{
338         dst[0]=src[0];
339         dst[1]=src[1];
340         dst[2]=src[2];
341         }
342         default:{
343         dst[0]+=(((src[0]-dst[0])*src[3])>>8);
344         dst[1]+=(((src[1]-dst[1])*src[3])>>8);
345         dst[2]+=(((src[2]-dst[2])*src[3])>>8);}
346         }
347             }}
348     return;
349 }
350 
351 if(image->color_type == FB_COLOR_ALPHA_8) /*lab3: font*/
352 {
353     //printf("you need implement fb_draw_image() FB_COLOR_ALPHA_8\n");
        exit(0);
354     for(iy=0;iy<h;iy++){
355 
356         for(ix=0;ix<w;ix++)
357             {
358 
359             dst = (char *)(LCD_MEM_BUFFER + (iy+y)*SCREEN_WIDTH + ix+x);
360             src = image->content + (iy)*image->line_byte + ix;
361             char pos0 = (char)color,
362                 pos1 = (char)(color >> 8),
363                 pos2 = (char)(color >> 16);
364             switch(src[0]){
365                 case 0:break;
366                 case 255:
367                     dst[0]=pos0;
368                     dst[1]=pos1;
369                     dst[2]=pos2;
370                     //break;
371                 default:
372                     dst[0]+=(((pos0-dst[0])*src[0])>>8);
373                     dst[1]+=(((pos1-dst[1])*src[0])>>8);
```

```
374                     dst[2]+=(((pos2-dst[2])*src[0])>>8);}
375                 }
376
377 }
378         return;
379     }
380 /*-------------------------------------------------------------*/
381     return;
382 }
383 /*
384 void color(int alpha,char* rsc,char* dst)
385 {
386     switch(alpha){
387         case 0:break;
388         case 255:
389             dst[0]=rsc[0];
390             dst[1]=rsc[1];
391             dst[2]=rsc[2];
392         default:
393             dst[0]+=(((rsc[0]-dst[0])*alpha)>>8);
394             dst[1]+=(((rsc[1]-dst[1])*alpha)>>8);
395             dst[2]+=(((rsc[2]-dst[2])*alpha)>>8);
396     }
397 }
398 */
399 /** draw a text string **/
400 void fb_draw_text(int x, int y, char *text, int font_size, int color)
401 {
402     fb_image *img;
403     fb_font_info info;
404     int i=0;
405     int len = strlen(text);
406     while(i < len)
407     {
408         img = fb_read_font_image(text+i, font_size, &info);
409         if(img == NULL) break;
410         fb_draw_image(x+info.left, y-info.top, img, color);
411         fb_free_image(img);
412
413         x += info.advance_x;
414         i += info.bytes;
415     }
416     return;
417 }
```

common/touch.c

```c
#include "common.h"

#include <stdio.h>
//#include <linux/input.h>
#include "input.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>

static struct finger_info{
   int x;
   int y;
   int event;
} infos[FINGER_NUM_MAX];
static int touch_fd;
static int cur_slot = 0;


/*return:
  TOUCH_NO_EVENT
  TOUCH_PRESS
  TOUCH_MOVE
  TOUCH_RELEASE
x: 0~1023
y: 0~599
finger: 0,1,2,3,4
*/
int touch_read1(int *x, int *y, int *finger)
{
   struct input_event data;
   int n, ret;
   if((n = read(touch_fd, &data, sizeof(data))) != sizeof(data)){
      printf("touch_read error %d, errno=%d\n", n, errno);
      return TOUCH_NO_EVENT;
   }
// printf("event read: type-code-value = %d-%d-%d\n", data.type, data.code,
    data.value);
   switch(data.type)
   {
   case EV_ABS:
      switch(data.code)
      {
      case ABS_MT_SLOT:
```

```
45          if(data.value >= 0 && data.value < FINGER_NUM_MAX){
46             int old = cur_slot;
47             cur_slot = data.value;
48             if(infos[old].event != TOUCH_NO_EVENT){
49                *x = infos[old].x;
50                *y = infos[old].y;
51                *finger = old;
52                ret = infos[old].event;
53                infos[old].event = TOUCH_NO_EVENT;
54                return ret;
55             }
56          }
57          break;
58       case ABS_MT_TRACKING_ID:
59          if(data.value == -1){
60             *x = infos[cur_slot].x;
61             *y = infos[cur_slot].y;
62             *finger = cur_slot;
63             infos[cur_slot].event = TOUCH_NO_EVENT;
64             return TOUCH_RELEASE;
65          }
66          else{
67             infos[cur_slot].event = TOUCH_PRESS;
68          }
69          break;
70       case ABS_MT_POSITION_X:
71          infos[cur_slot].x = data.value;
72          if(infos[cur_slot].event != TOUCH_PRESS){
73                infos[cur_slot].event = TOUCH_MOVE;
74             }
75             break;
76          case ABS_MT_POSITION_Y:
77             infos[cur_slot].y = data.value;
78             if(infos[cur_slot].event != TOUCH_PRESS){
79                infos[cur_slot].event = TOUCH_MOVE;
80             }
81             break;
82       }
83       break;
84    case EV_SYN:
85       switch(data.code)
86       {
87       case SYN_REPORT:
88          if(infos[cur_slot].event != TOUCH_NO_EVENT){
89             *x = infos[cur_slot].x;
90             *y = infos[cur_slot].y;
```

```
91              *finger = cur_slot;
92              ret = infos[cur_slot].event;
93              infos[cur_slot].event = TOUCH_NO_EVENT;
94              return ret;
95          }
96          break;
97       }
98       break;
99    }
100   return TOUCH_NO_EVENT;
101 }
102
103 #define BTN_POS_X 10
104 #define BTN_POS_Y 10
105 #define BTN_W 40
106 #define BTN_H 40
107 #define LINE_COLOR FB_COLOR(255,100,100)
108 #define BTN_COLOR FB_COLOR(255,255,0)
109 struct rlib_pos {int x, y;};
110 void draw_btn() {
111    const int padding = 5;
112    fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,FB_COLOR(255,255,255));
113    fb_draw_rect(BTN_POS_X, BTN_POS_Y, BTN_W, BTN_H, BTN_COLOR);
114    fb_draw_text(BTN_POS_X+padding, BTN_POS_Y+padding, "clear", 20, BTN_COLOR);
115 }
116 bool pos_in_btn(int x, int y) {
117    return x > BTN_POS_X && x < BTN_POS_X + BTN_W && y > BTN_POS_Y && y <
        BTN_POS_Y + BTN_H;
118 }
119 void on_touch_read_finish(int x, int y, int finger, int eventId) {
120    static struct rlib_pos history_poss[FINGER_NUM_MAX];
121    if(!(finger < FINGER_NUM_MAX && finger >= 0))
122       return;
123    printf("DEBUG: finger=%d\n", finger);
124    switch(eventId) {
125       case TOUCH_PRESS:
126          if(pos_in_btn(x, y)) {
127             draw_btn();
128          }
129          else {
130             history_poss[finger].x = x;
131             history_poss[finger].y = y;
132             fb_draw_pixel(x, y, LINE_COLOR);
133          }
134          break;
135       case TOUCH_MOVE:
```

```
136             fb_draw_line(x, y, history_poss[finger].x, history_poss[finger].y,
                    LINE_COLOR);
137             history_poss[finger].x = x;
138             history_poss[finger].y = y;
139             break;
140         }
141
142     }
143     int touch_read(int *px, int *py, int *pfinger) {
144         int x,y,finger;
145         int type = touch_read1(&x, &y, &finger);
146         on_touch_read_finish(x, y, finger, type); // bug
147         fb_update();
148         *px = x; *py = y; *pfinger = finger;
149         return type;
150     }
151
152     void touch_init(char *dev)
153     {
154         touch_fd = open(dev, O_RDONLY);
155         if(touch_fd < 0){
156             printf("touch_init open %s error!errno = %d\n", dev, errno);
157             return;
158         }
159         font_init("/data/local/font.ttc");
160         draw_btn();
161         fb_update();
162         return;
163     }
```

lab1/main.c

```
1   #include <stdio.h>
2
3   int main(int argc, char* argv[])
4   {
5       printf("Hello embedded linux!\n");
6       return 0;
7   }
```

lab2/main.c

```
1   #include <sys/mman.h>
2   #include <linux/fb.h>
3   #include <stdio.h>
4
5   #include "../common/common.h"
```

```
6
7  #define RED  FB_COLOR(255,0,0)
8  #define ORANGE FB_COLOR(255,165,0)
9  #define YELLOW FB_COLOR(255,255,0)
10 #define GREEN FB_COLOR(0,255,0)
11 #define CYAN FB_COLOR(0,127,255)
12 #define BLUE FB_COLOR(0,0,255)
13 #define PURPLE FB_COLOR(139,0,255)
14 #define WHITE FB_COLOR(255,255,255)
15 #define BLACK FB_COLOR(0,0,0)
16
17 int color[9] = {RED,ORANGE,YELLOW,GREEN,CYAN,BLUE,PURPLE,WHITE,BLACK};
18
19 int main(int argc, char* argv[])
20 {
21    int row,column,i;
22    int32_t start, end;
23
24    fb_init("/dev/graphics/fb0");
25    fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
26    fb_update();
27    printf("\n========== Start Test ==========\n");
28
29    sleep(1);
30    start = fb_get_time();
31    for(row=-5; row<605; row+=2)
32    {
33       for(column=-5; column<1029; column+=2)
34       {
35          fb_draw_pixel(column,row,YELLOW);
36          fb_update();
37       }
38    }
39    end = fb_get_time();
40    printf("draw pixel: %d ms\n",end - start);
41
42    sleep(1);
43    start = fb_get_time();
44    for(row=-35,i=0; row<635; row+=35)
45    {
46       for(column=-25; column<1050; column+=25)
47       {
48          fb_draw_rect(column,row,20,20,color[++i%9]);
49          fb_update();
50       }
51    }
```

```
52      end = fb_get_time();
53      printf("draw rect: %d ms\n",end - start);
54
55      sleep(1);
56      fb_draw_rect(300,200,400,200,BLACK);
57      fb_update();
58      start = fb_get_time();
59      for(row=0;row<=400;row+=20){
60          fb_draw_line(500-300,300-200+row,500+300,300+200-row,color[1]);
61          fb_update();
62      }
63      for(column=0;column<=400;column+=20){
64          fb_draw_line(500-200+column,300-200,500+200-column,300+200,color[2]);
65          fb_update();
66      }
67      end = fb_get_time();
68      printf("draw line: %d ms\n", end - start);
69      return 0;
70  }
```

lab3/main.c

```
1   #include <stdio.h>
2   #include "../common/common.h"
3
4   #define RED  FB_COLOR(255,0,0)
5   #define ORANGE FB_COLOR(255,165,0)
6   #define YELLOW FB_COLOR(255,255,0)
7   #define GREEN FB_COLOR(0,255,0)
8   #define CYAN FB_COLOR(0,127,255)
9   #define BLUE FB_COLOR(0,0,255)
10  #define PURPLE FB_COLOR(139,0,255)
11  #define WHITE FB_COLOR(255,255,255)
12  #define BLACK FB_COLOR(0,0,0)
13
14  int* main(int argc, char *argv[])
15  {
16      fb_init("/dev/graphics/fb0");
17      font_init("/data/local/font.ttc");
18
19      fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
20      fb_update();
21
22      fb_image *img;
23      img = fb_read_jpeg_image("/data/local/jpg_test.jpg");
24      fb_draw_image(0,0,img,0);
```

```
25      fb_update();
26      fb_free_image(img);
27
28      img = fb_read_png_image("/data/local/png_test.png");
29      fb_draw_image(100,300,img,0);
30      fb_update();
31      fb_free_image(img);
32
33      img = fb_read_font_image("嵌",30,NULL);
34      fb_draw_image(400,350,img,RED);
35      fb_update();
36      fb_free_image(img);
37
38      fb_draw_text(50,50,"床前明月光，疑是地上霜。",64,PURPLE);
39      fb_draw_text(50,120,"举头望明月，低头思故乡。",64,PURPLE);
40      fb_update();
41      return 0;
42  }
```

lab4/main.c

```
1   #include <stdio.h>
2   #include "../common/common.h"
3
4   int main(int argc, char *argv[])
5   {
6       fb_init("/dev/graphics/fb0");
7       fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,FB_COLOR(255,255,255));
8       fb_update();
9
10      touch_init("/dev/input/event3");
11
12      int type,x,y,finger,i;
13      while(1){
14          type = touch_read(&x,&y,&finger);
15          switch(type){
16              case TOUCH_PRESS:
17                  printf("TOUCH_PRESS: x=%d,y=%d,finger=%d\n",x,y,finger);
18                  break;
19              case TOUCH_MOVE:
20                  printf("TOUCH_MOVE: x=%d,y=%d,finger=%d\n",x,y,finger);
21                  break;
22              case TOUCH_RELEASE:
23                  printf("TOUCH_RELEASE: x=%d,y=%d,finger=%d\n",x,y,finger);
24                  break;
25              default:
```

```
26              break;
27          }
28      }
29      return 0;
30
31  }
```

lab5/main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <errno.h>
4
5   #include <sys/types.h>
6   #include <sys/stat.h>
7   #include <fcntl.h>
8
9   #include <pthread.h>
10  #include <sys/ioctl.h>
11
12  #include "../common/common.h"
13
14  /*====================================================================*/
15
16  #define LED_IOC_MAGIC 'L'
17  #define LED_ON _IO(LED_IOC_MAGIC, 0)
18  #define LED_OFF _IO(LED_IOC_MAGIC, 1)
19
20  static int led_fd = -1;
21  void led_set(int on)
22  {
23    on? ioctl(led_fd,LED_ON) : ioctl(led_fd,LED_OFF);
24     return;
25  }
26
27  /*====================================================================*/
28
29  int frequency;
30
31  const int trackX = 212, trackY=300, popupY=270;
32  inline void clear_screen();
33  int threading(void * m);
34  const int refreshX = 240, refreshY = 240, refreshW = 800, refreshH = 120;
35  fb_image *backgroundImage, *refreshBackground;
36
37  int main(int argc, char* argv[])
```

```
38 {
39     fb_init("/dev/graphics/fb0");
40     font_init("/data/local/font.ttc");
41     touch_init("/dev/input/event3");
42 
43     if(led_fd == -1)
44     {
45         led_fd = open("/dev/led", O_RDWR);
46         if(led_fd < 0){
47             printf("open /dev/led failed, errno = %d\n", errno);
48         }
49     }
50 
51     // create thread
52     pthread_t t1;
53     int i,ret;
54     ret = pthread_create(&t1,NULL,threading,NULL);
55     if(ret≠0)
56     {
57         printf("Create pthread error!\n");
58         exit(1);
59     }
60 
61 
62     // draw background image
63     backgroundImage = fb_read_jpeg_image("/background.jpg");
64     refreshBackground = fb_get_sub_image(
65           backgroundImage, refreshX, refreshY, refreshW, refreshH);
66     clear_screen();
67 
68     // draw controller
69     fb_draw_track(trackX, trackY);
70     fb_draw_infill(trackX, trackY, 0);
71     fb_draw_slider(trackX, trackY);
72     // fb_draw_popup(x, y);
73     // fb_draw_poptext(x, y, val);
74 
75     fb_update();
76 
77     int type, x, y, finger;
78     int isPressed, lastSliderX;
79 
80     while(1){
81         type = touch_read(&x, &y, &finger);
82         switch (type) {
83             case TOUCH_PRESS:
```

```
84              printf("TOUCH_PRESS: x=%d,y=%d,finger=%d\n", x, y, finger);
85              if( x > lastSliderX - 25 && x < lastSliderX + 25 &&
86                 y > trackY - 25 && y < trackY + 25 ){
87                 fb_draw_popup(lastSliderX, popupY);
88                 fb_draw_poptext(lastSliderX, popupY, (lastSliderX - trackX)/6);
89                 fb_update();
90                 isPressed = 1;
91              }
92              break;
93           case TOUCH_MOVE: {
94              printf("TOUCH_MOVE: x=%d,y=%d,finger=%d\n", x, y, finger);
95              if (x < trackX)
96                 x = trackX;
97              if (x > trackX + 600)
98                 x = trackX + 600;
99              if (isPressed){
100                //redraw image
101                //fb_draw_image(refreshX, refreshY, refreshBackground, 0);
102                fb_draw_image(0, 0, backgroundImage, 0);
103                fb_draw_track(trackX, trackY);
104                fb_draw_infill(trackX, trackY, x - trackX);
105                fb_draw_slider(x, trackY);
106                fb_draw_popup(x, popupY);
107                fb_draw_poptext(x, popupY, (x - trackX)/6);
108                lastSliderX = x;
109                fb_update();
110
111                //get frequency
112                frequency = (x - trackX) / 6;
113             }
114             break;
115          }
116          case TOUCH_RELEASE:
117             printf("TOUCH_RELEASE: x=%d,y=%d,finger=%d\n", x, y, finger);
118             if (x < trackX)
119                x = trackX;
120             if (x > trackX + 600)
121                x = trackX + 600;
122             if (isPressed){
123                // redraw image, clear popup and popup text
124                //fb_draw_image(refreshX, refreshY, refreshBackground, 0);
125                fb_draw_image(0, 0, backgroundImage, 0);
126                fb_draw_track(trackX, trackY);
127                fb_draw_infill(trackX, trackY, x - trackX);
128                fb_draw_slider(x, trackY);
129                fb_update();
```

```
130                 lastSliderX = x;
131                 isPressed = 0;
132              }
133              break;
134           default:
135              printf("unknown\n");
136              break;
137        }
138     }
139
140     return 0;
141  }
142
143  void clear_screen(){
144     fb_draw_image(0, 0, backgroundImage, 0);
145     fb_draw_image(974, 550, fb_read_png_image("/button_50.png"), 0);
146     //fb_draw_rect(0,0,1024,600,0);
147     fb_update();
148  }
149
150  int threading(void *m)
151  {
152     int cmd;
153     while(1)
154     {
155        if(frequency == 0){
156           printf("-- 1\n");
157           led_set(0);
158        } else if (frequency == 100) {
159           printf("-- 2\n");
160           led_set(1);
161        } else {
162           printf("-- 3, %d\n", 10000000 / (frequency + 100));
163           led_set(1);
164           usleep(10000000 / (frequency + 50));
165           led_set(0);
166           usleep(10000000 / (frequency + 50));
167        }
168     }
169  }
```