華中科技大學

操作系统课程设计报告

学 生 姓 名: 胡思勖

学 号: U201514898

专 业: 计算机科学与技术

班 级: 计卓 1501

指 导 教 师: 张杰

2018-03-03

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

首先确定 common/rules.mk 文件配置的编译器目录正确: DIR:=" ../../tools/android-ndk-r8c"。

1. 编译生成 lab1:
   - make
2. 把文件上传到实验板上的/data 目录:
   - adb push lab1 /data/local/
   - adb shell chmod +x /data/local/lab1
3. 登录到实验板上运行:
   - adb shell
   - cd /data/local
   - ./lab1
4. 或者直接运行程序:
   - adb shell /data/local/lab1

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

画点函数老师已经在代码中给出。画线函数和画矩形函数则只需要调用画点函数即可。

2.3.1 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。该函数的具体思路如图2.1所示。

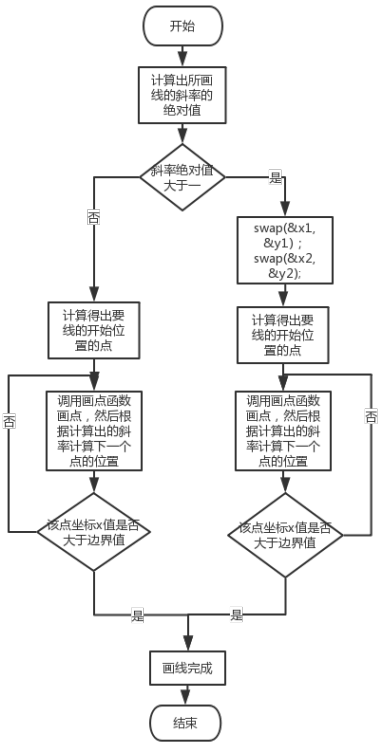

图 2.1: 画线函数流程

2.3.2 画矩形函数

矩形函数的主要思路为：首先矫正所要画矩形的起始点的坐标值，不能小于零，若小于零，则将 x 赋值为 0，然后宽度的值 w 赋值为 w+x，同样对 y 坐标的处理也一样，这样所得到的矩形为所画矩形或者所画矩形的一部分。然后调用 update () 函数，最后使用两个 for 循环，每次画完一个点以后，然后计算得到所画矩形的下一个点的位置，然后调用画点函数进行画点，直到画完为止。

在本次实验过程中，我们考虑了对画线函数进行优化，即使用超级采样 super-Sampling 消除锯齿算法对所画出线进行抗锯齿处理。超级采样简单的说就对每一个被采样的象素进行更细分的采样和计算。其方法和我们前面所介绍的类似。这一做法非常的占用资源因为说要计算的数据比起原来成倍的增加。可以理解为这种工作状态实际上相当于在进行更高分辨率的运算之后将画面用低分辨率来显示。但是很遗憾，在测试的过程中得到的结果却是很不尽人意。最后还是采用了常规的解法。

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3.1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。

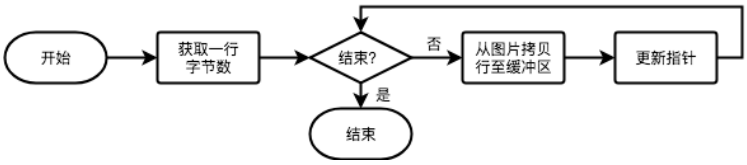

图 3.1: Jpeg 显示图片流程

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。

在实现时遇到了程序 SegmentFault 的错误，经过长时间排查，最终确定了原因是：

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。

添加边界检查代码以及去除 Makefile 中-O3 的编译参数后，程序正确运行。在实现时遇到了颜色显示错误的问题，经过排查，最终确定了原因是：

1. 混淆了 A、R、G、B 四个通道字节在内存中的低位和高位顺序。
2. 修改了指针指向字节的顺序后，颜色恢复正常。

显示 PNG 图片的流程如图3.2所示。

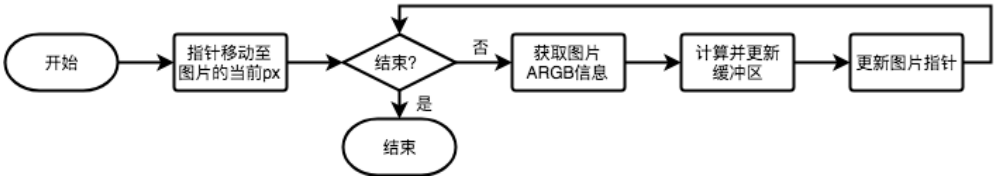

图 3.2: Png 显示图片流程

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. 每个手指轨迹的颜色不同
2. 轨迹是连贯的，不能断断续续
3. 轨迹要比较粗，线宽不能是 1 个点
4. 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、具有一定宽度的线。

其模块间调用关系如图4.1所示。

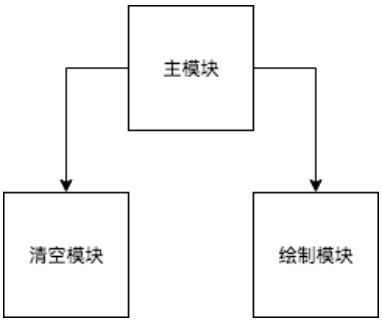

图 4.1: 模块关系

4.3 实验过程

4.3.1 清空模块

清空模块实现比较简单。只需要向外部提供一个 clear_screen 函数即可。该函数用 fb_draw_image 重绘屏幕和按钮并更新 FrameBuffer。

4.3.2 绘制模块

绘制模块通过在两个点连线的每一个像素上绘制一定大小和颜色的实心圆来实现绘制连贯、有一定宽度的线的功能。这种实现的主要优点是简单易实现，缺点是进行了大量的重复绘制（两个间隔 1px 圆存在大量重合的部分）。更好的实现是计算切线，相应地绘制两个半圆和一个矩形。出于时间和难度等原因，未采用这种实现。

绘制模块总共实现了 fb_draw_circle 和 fb_draw_thick_line 两个函数，其中 fb_draw_circle 函数负责在指定点绘制指定大小、颜色的实心圆，fb_draw_thick_line 负责在两点连线的每一个像素上调用 fb_draw_circle 函数以画线。

fb_draw_circle 函数在指定正方形区域内遍历每一个点（像素）是否在圆内（即到圆心的距离是否小于等于半径）来决定是否绘制该像素。fb_draw_circle 的流程如图4.2所示。

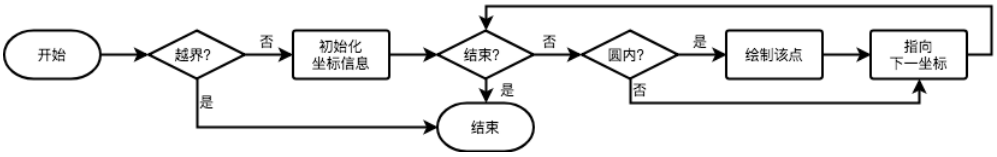

图 4.2: fb_draw_circle 流程

fb_draw_thick_line 同实验二中的画线函数类似。在最初的实现中，当线过于陡峭时，会出现线不连贯的情况。经过排查是因为最初的实现通过 x 计算点 y，当斜率大于 1 时，即使是很小的 x 的变化也会引起 y 的很大变化。导致两个点距离过大，画出的圆无法连接成线。

最终实现的 fb_draw_thick_line 通过检查斜率与 1 的关系来决定是按照 x（斜率小于 1）还是 y（斜率大于 1）来计算每一个点 (x, y)。并在该点上调用 fb_draw_circle 函数。fb_draw_thick_line 的流程如图4.3所示。

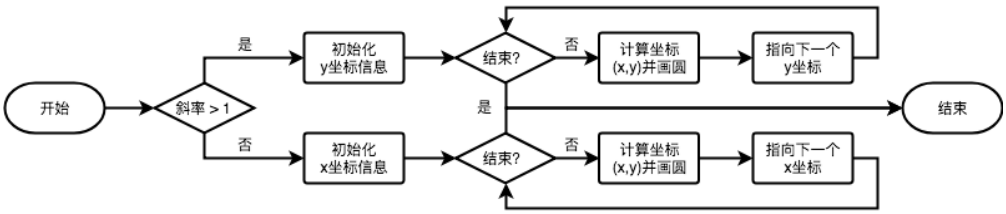

图 4.3: fb_draw_thick_line 流程

在测试时，出现了画出的线无法清空以及屏幕叠加的问题，经过长时间排查，最终确定了原因是：

1. 之前执行的程序没有退出（因为主模块是一个无限循环）而出现了同时执行若干个程序的情况。

   通过 kill 命令杀死残留进场后，程序行为恢复正常。

4.3.3 主模块

如实验设计所述，主模块的本质是一个无限循环，在循环内部轮询以获取触摸事件。主模块需要保存 5 个不同触摸点的历史坐标信息以及新读取到的坐标、触摸点信息。主模块对于三种不同的触摸事件，需要作出如下的响应：

1. TOUCH_MOVE：主模块需要获取触摸点编号（0-4），并根据编号设置相应的颜色、宽度（用于区分不同的触摸点），并且绘制一条从该触摸点的上一个坐标到当前坐标的直线，并更新该触摸点的历史坐标。

2. TOUCH_PRESS：主模块需要记录该次触摸的坐标以及触摸点信息，并更新该触摸点的历史坐标。

3. TOUCH_RELEASE：主模块需要记录该次释放的坐标即触摸点信息，并和 TOUCH_PRESS 中记录的信息进行比对。如果 TOUCH_PRESS 和 TOUCH_RELEASE 事件的坐标都在清空按钮的坐标范围内，并且是同一种触摸点的话，则证明是按下了清空按钮。需要执行清空操作，并更新该触摸点的历史坐标。

主模块的流程如图4.4所示。

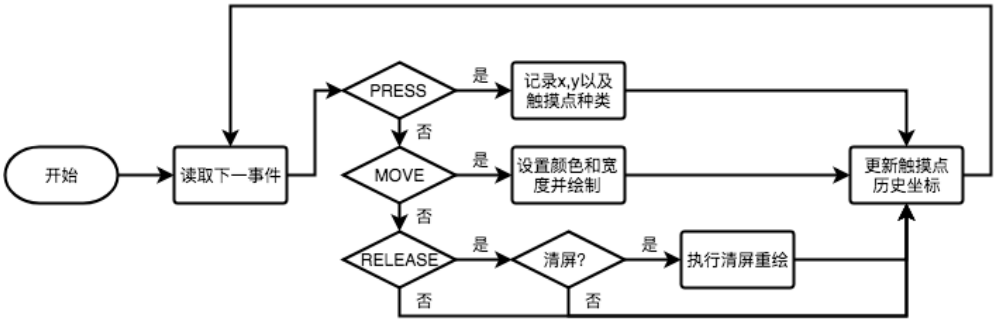

图 4.4: 主模块流程

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。

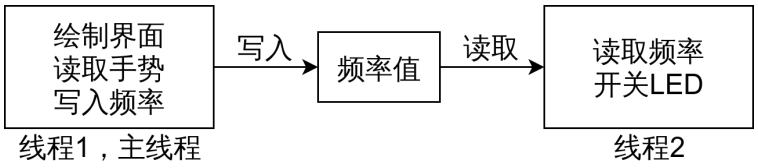

图 5.1: 线程之间的关系

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5.2所示，图层越高的控件越后绘制。

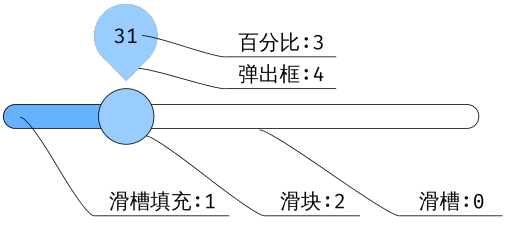

图 5.2: 控件示意图

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5.3所示。

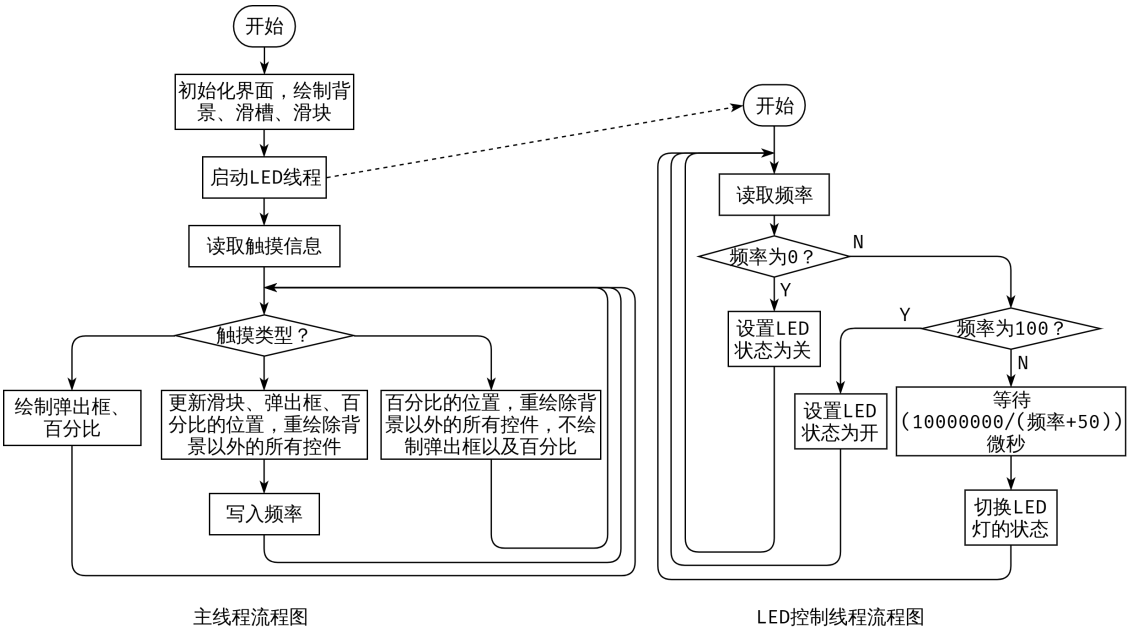

图 5.3: 两个线程的流程图

5.3 实验过程

在程序编写完成后，首先进行编译，然后通过 adb 上传开发板后运行。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。前些时自行编译并烧录过 $\mu - COS$ 内核，与这一实验的 linux 内核相对比，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、evnet 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。在实验之后我甚至成功的将实验 1 ~ 3 的内容移植到 x86 linux 平台运行，这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>
#include <math.h>

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

```

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

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

```
104             update_area.x2 = x2;
105         if (update_area.y2 < y2)
106             update_area.y2 = y2;
107     }
108     return;
109 }
110 
111 /*====================================================================*/
112 
113 void fb_draw_pixel(int x, int y, int color) {
114     if (x < 0 || y < 0 || x >= SCREEN_WIDTH || y >= SCREEN_HEIGHT)
115         return;
116     _update_area(x, y, 1, 1);
117     int *tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
118     *tmp = color;
119     return;
120 }
121 
122 void fb_draw_rect(int x, int y, int w, int h, int color) {
123     int i;
124     int j;
125     if (x < 0) {
126         w += x;
127         x = 0;
128     }
129     if (x + w > SCREEN_WIDTH) {
130         w = SCREEN_WIDTH - x;
131     }
132     if (y < 0) {
133         h += y;
134         y = 0;
135     }
136     if (y + h > SCREEN_HEIGHT) {
137         h = SCREEN_HEIGHT - y;
138     }
139     if (w <= 0 || h <= 0)
140         return;
141     _update_area(x, y, w, h);
142     /*-----------------------------------------------------*/
143     // printf("you need implement fb_draw_rect()\n"); exit(0);
144     for (i = y ; i < y + h ; ++i) {
145         for (j = x ; j < x + w ; ++j) {
146             fb_draw_pixel(j, i, color);
147         }
148     }
149     /*-----------------------------------------------------*/
```

```
150     return;
151 }
152
153 void swap(int *a, int *b) {
154     int temp = *a;
155     *a = *b;
156     *b = temp;
157 }
158
159 // returns absolute value of number
160 float absolute(float x ) {
161     return x > 0 ? x : -x;
162 }
163
164 //returns integer part of a floating point number
165 int iPartOfNumber(float x) {
166     return (int)x;
167 }
168
169 //rounds off a number
170 int roundNumber(float x) {
171     return iPartOfNumber(x + 0.5) ;
172 }
173
174 //returns fractional part of a number
175 float fPartOfNumber(float x) {
176     return x > 0 ? x - iPartOfNumber(x) : x - (iPartOfNumber(x) + 1);
177 }
178
179 //returns 1 - fractional part of number
180 float rfPartOfNumber(float x) {
181     return 1 - fPartOfNumber(x);
182 }
183
184 // get new color according to current color and brightness
185 int getNewColor(float bri, int x) {
186     return x;
187 //    return FB_COLOR( ((int)(bri * 0.2990 * ((x & 0xff0000) >> 16))),\
188 //        ((int)(bri * 0.5870 * ((x & 0xff00) >> 8))),\
189 //        ((int)(bri * 0.1140 * (x & 0xff))) );
190 }
191
192 void fb_draw_line(int x1, int y1, int x2, int y2, int color) {
193     /*-------------------------------------------------*/
194     // printf("you need implement fb_draw_line()\n"); exit(0);
195     int steep = absolute(y2 - y1) > absolute(x2 - x1) ;
```

```
196
197     // swap the co-ordinates if slope > 1 or we
198     // draw backwards
199     if (steep) {
200         swap(&x1, &y1);
201         swap(&x2, &y2);
202     }
203     if (x1 > x2) {
204         swap(&x1, &x2);
205         swap(&y1, &y2);
206     }
207
208     //compute the slope
209     float dx = x2 - x1;
210     float dy = y2 - y1;
211     float gradient = dy / dx;
212     if (dx == 0.0)
213         gradient = 1;
214
215     int xpxl1 = x1;
216     int xpxl2 = x2;
217     float intersectY = y1;
218
219     // main loop
220     if (steep) {
221         int x;
222         for (x = xpxl1 ; x <= xpxl2 ; x++) {
223             // pixel coverage is determined by fractional
224             // part of y co-ordinate
225             fb_draw_pixel(iPartOfNumber(intersectY), x, getNewColor(intersectY,
                    color));
226             fb_draw_pixel(iPartOfNumber(intersectY) - 1, x, getNewColor(
                   intersectY, color));
227             intersectY += gradient;
228         }
229     } else {
230         int x;
231         for (x = xpxl1 ; x <= xpxl2 ; x++) {
232             // pixel coverage is determined by fractional part of y co-ordinate
233             fb_draw_pixel(x, iPartOfNumber(intersectY), getNewColor(intersectY,
                    color));
234             fb_draw_pixel(x, iPartOfNumber(intersectY) - 1, getNewColor(
                   intersectY, color));
235             intersectY += gradient;
236         }
237     }
```

```
238     /*-----------------------------------------------*/
239     return;
240 }
241 
242 /*=================================================================*/
243 
244 void fb_draw_image(int x, int y, fb_image *image, int color) {
245     if (image == NULL)
246         return;
247 
248     int ix = 0; //image x
249     int iy = 0; //image y
250     int w = image->pixel_w; //draw width
251     int h = image->pixel_h; //draw height
252 
253     if (x < 0) {
254         w += x;
255         ix -= x;
256         x = 0;
257     }
258     if (y < 0) {
259         h += y;
260         iy -= y;
261         y = 0;
262     }
263 
264     if (x + w > SCREEN_WIDTH) {
265         w = SCREEN_WIDTH - x;
266     }
267     if (y + h > SCREEN_HEIGHT) {
268         h = SCREEN_HEIGHT - y;
269     }
270     if ((w <= 0) || (h <= 0))
271         return;
272 
273     _update_area(x, y, w, h);
274 
275     int *dst = (int *)(LCD_MEM_BUFFER + y * SCREEN_WIDTH + x);
276     char *src = image->content + iy * image->line_byte + ix * 4;
277     /*-------------------------------------------------------------*/
278 
279     int temp;
280     int ww, i;
281 
282     if (image->color_type == FB_COLOR_RGB_8880) { /*lab3: jpg*/
283         //printf("you need implement fb_draw_image() FB_COLOR_RGB_8880\n");
```

```
284         //exit(0);
285         int bytes = (image->line_byte >= 4 * w) ? w * 4 : image->line_byte;
286         for(ww = 0; ww < h; ++ww){
287             memcpy(dst, src, bytes);
288             dst += SCREEN_WIDTH;
289             src += image->line_byte;
290         }
291         // int i, j;
292         // for (i = y ; i < y + h ; ++i) {
293         //     for (j = x ; j < x + w ; ++j) {
294         //         int * temp = LCD_MEM_BUFFER + i * SCREEN_WIDTH + j;
295         //         char * pcolor = (image->content + (i - y) * image->line_byte
                        + (j - x) * 4);
296         //         *temp = *(int *)pcolor;
297         //     }
298         // }
299         return;
300     }
301 
302     if (image->color_type == FB_COLOR_RGBA_8888) { /*lab3: png*/
303         //printf("you need implement fb_draw_image() FB_COLOR_RGBA_8888\n");
304         //exit(0);
305         int i, j;
306         char B1, G1, R1, alpha;
307         for (i = y ; i < y + h ; ++i) {
308             for (j = x ; j < x + w ; ++j) {
309                 int * temp = (int *)LCD_MEM_BUFFER + i * SCREEN_WIDTH + j;
310 
311                 char * pcolor = image->content + (i - y) * image->line_byte + (
                        j - x) * 4;
312                 char * p = temp;
313 
314                 alpha = *(char *)(pcolor + 3);
315                 R1 = *(char *)(pcolor + 2);
316                 G1 = *(char *)(pcolor + 1);
317                 B1 = *(char *)(pcolor);
318                 // printf("%d %d %d %d\t", alpha, R1, G1, B1);
319                 switch (alpha) {
320                     case 0: break;
321                     case 255: p[0] = B1; p[1] = G1; p[2] = R1; break;
322                     default:
323                         // p[0] = (255 - alpha) * p[0] + alpha * B1;
324                         // p[1] = (255 - alpha) * p[1] + alpha * G1;
325                         // p[2] = (255 - alpha) * p[2] + alpha * R1;
326                         p[0] += (((B1 - p[0]) * alpha) >> 8);
327                         p[1] += (((G1 - p[1]) * alpha) >> 8);
```

```
328                         p[2] += (((R1 - p[2]) * alpha) >> 8);
329                         break;
330                 }
331             }
332         }
333         return;
334     }
335 
336     if (image->color_type == FB_COLOR_ALPHA_8) { /*lab3: font*/
337         //printf("you need implement fb_draw_image() FB_COLOR_ALPHA_8\n");
338         //exit(0);
339         char *i;
340         int *j;
341         int count;
342         int R1, G1, B1, alpha;
343         for(ww = 0; ww < h; ++ww){
344             i = src;
345             j = dst;
346 
347            for(count = 0; count < w; ++count){
348                 alpha = *(char *)(i);
349                 R1    = (color & 0xff0000) >> 16;
350                 G1    = (color & 0xff00) >> 8;
351                 B1    = (color & 0xff);
352 
353                 *((char *)(j)    ) += (((B1 - *((char *)(j)    )) * alpha) >>
                        8);
354                 *((char *)(j) + 1) += (((G1 - *((char *)(j) + 1)) * alpha) >>
                        8);
355                 *((char *)(j) + 2) += (((R1 - *((char *)(j) + 2)) * alpha) >>
                        8);
356 
357                 ++i;
358                 ++j;
359             }
360             dst += SCREEN_WIDTH;
361             src += image->line_byte;
362         }
363         return;
364     }
365     /*--------------------------------------------------------------*/
366     return;
367 }
368 
369 /** draw a text string **/
370 void fb_draw_text(int x, int y, char *text, int font_size, int color) {
```

```
371     fb_image *img;
372     fb_font_info info;
373     int i = 0;
374     int len = strlen(text);
375     while (i < len) {
376         img = fb_read_font_image(text + i, font_size, &info);
377         if (img == NULL)
378             break;
379         fb_draw_image(x + info.left, y - info.top, img, color);
380         fb_free_image(img);
381 
382         x += info.advance_x;
383         i += info.bytes;
384     }
385     return;
386 }
387 
388 void fb_draw_circle(int x, int y, int r, int color) {
389     if (x - r < 0 || y - r < 0 || x + r > SCREEN_WIDTH || y + r > SCREEN_HEIGHT
        ) return;
390     _update_area(x - r, y - r, 2 * r , 2 * r);
391     int i, j;
392     for (i = y - r ; i <= y + r ; ++i) {
393         for (j = x - r ; j < x + r ; ++j) {
394             if ((i - y) * (i - y) + (j - x) * (j - x) <= r * r) {
395                 fb_draw_pixel(j, i, color);
396             }
397         }
398     }
399 }
400 
401 void fb_draw_thick_line(int x1, int y1, int x2, int y2, int radius, int color)
    {
402     int i;
403     int steep = absolute(y2 - y1) > absolute(x2 - x1);
404     if (steep) {
405         double slope = (double)(x2 - x1) / (y2 - y1);
406 
407         if (y2 < y1) {
408             swap(&y1, &y2);
409             swap(&x1, &x2);
410         }
411         for (i = y1 ; i <= y2 ; ++i) {
412             fb_draw_circle((int)((double)x1 + (double)(slope * (i - y1))), i,
                radius, color);
413         }
```

```
414     } else {
415         double slope = (double)(y2 - y1) / (x2 - x1);
416         if (x2 < x1) {
417             swap(&x1, &x2);
418             swap(&y1, &y2);
419         }
420         // int y = (y2 > y1) ? y1 : y2;
421         // int y_diff    = (y2 > y1) ? (y2 - y1) : (y1 - y2);
422         // _update_area(x1 - radius, y - radius, x2 - x1 + 2 * radius, y_diff +
                2 * radius);
423         for (i = x1 ; i <= x2 ; ++i) {
424             fb_draw_circle(i, (int)((double)y1 + (double)(slope * (i - x1))),
                    radius, color);
425         }
426     }
427 }
428
429 #define FB_SLIDER_WIDTH 600
430 #define FB_SLIDER_HEIGHT 20
431 #define FB_SLIDER_RADIUS 20
432 #define FB_SLIDER_BACKGROUND_COLOR FB_COLOR(87, 96, 111)
433 #define FB_SLIDER_FOREGROUND_COLOR FB_COLOR(241, 242, 246)
434 #define FB_SLIDER_FILL_COLOR FB_COLOR(112, 161, 255)
435 #define FB_SLIDER_STROKE_LINE_WIDTH 1
436 #define FB_SLIDER_POPUP_RADIUS 25
437 #define FB_SLIDER_POPUP_COLOR FB_COLOR(55, 66, 250)
438 #define FB_SLIDER_TEXT_COLOR FB_COLOR(241 , 242, 246)
439 #define FB_HALF_DIAGONAL ((double)FB_SLIDER_POPUP_RADIUS / sqrt(2.0))
440
441 void fb_draw_track(int x, int y) {
442     fb_draw_circle(x, y, FB_SLIDER_HEIGHT / 2, FB_SLIDER_BACKGROUND_COLOR);
443     fb_draw_circle(x, y, FB_SLIDER_HEIGHT / 2 - FB_SLIDER_STROKE_LINE_WIDTH,
            FB_SLIDER_FILL_COLOR);
444
445     fb_draw_circle(x + FB_SLIDER_WIDTH, y, FB_SLIDER_HEIGHT / 2,
            FB_SLIDER_BACKGROUND_COLOR);
446     fb_draw_circle(x + FB_SLIDER_WIDTH, y, FB_SLIDER_HEIGHT / 2 -
            FB_SLIDER_STROKE_LINE_WIDTH, FB_SLIDER_FOREGROUND_COLOR);
447
448     fb_draw_rect(x, y - FB_SLIDER_HEIGHT / 2, FB_SLIDER_WIDTH, FB_SLIDER_HEIGHT
            , FB_SLIDER_BACKGROUND_COLOR);
449     fb_draw_rect(x, y - FB_SLIDER_HEIGHT / 2 + FB_SLIDER_STROKE_LINE_WIDTH,
            FB_SLIDER_WIDTH, FB_SLIDER_HEIGHT - 2 * FB_SLIDER_STROKE_LINE_WIDTH,
            FB_SLIDER_FOREGROUND_COLOR);
450 }
451
```

```
452 void fb_draw_infill(x, y, length) {
453     if (length <= 0 || length > FB_SLIDER_WIDTH) {
454         return;
455     }
456     fb_draw_rect(x, y - FB_SLIDER_HEIGHT / 2 + FB_SLIDER_STROKE_LINE_WIDTH,
            length, FB_SLIDER_HEIGHT - 2 * FB_SLIDER_STROKE_LINE_WIDTH,
            FB_SLIDER_FILL_COLOR);
457 }
458
459 void fb_draw_slider(x, y) {
460     fb_draw_circle(x, y, FB_SLIDER_RADIUS, FB_SLIDER_BACKGROUND_COLOR);
461     fb_draw_circle(x, y, FB_SLIDER_RADIUS - FB_SLIDER_STROKE_LINE_WIDTH,
            FB_COLOR(30, 144, 255));
462 }
463
464 void fb_draw_popup(x, y) {
465     // draw the below half triangle
466     int i, j;
467     for (i = y ; i >= y - FB_HALF_DIAGONAL ; --i) {
468         for (j = x - (y - i) ; j <= x + (y - i) ; ++j) {
469             fb_draw_pixel(j, i, FB_SLIDER_POPUP_COLOR);
470         }
471     }
472
473     // draw the up half circle
474     fb_draw_circle(x, y - 2 * FB_HALF_DIAGONAL , FB_SLIDER_POPUP_RADIUS,
            FB_SLIDER_POPUP_COLOR);
475 }
476
477 void fb_draw_poptext(int x, int y, int value) {
478     if (value < 0 || value > 100) return;
479
480     char number[3];
481     memset(number, 0, 3);
482     if (value == 100) {
483         sprintf(number, "ON");
484     } else {
485         if (value < 10) {
486             number[0] = '0';
487             number[1] = '0' + value;
488         } else {
489             number[0] = '0' + value / 10;
490             number[1] = '0' + value % 10;
491         }
492     }
493
```

```
494     fb_draw_text(x - 15, y - 1.5 * (double)FB_HALF_DIAGONAL , number, 30,
            FB_SLIDER_TEXT_COLOR);
495 }
```

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  inline void clear_screen();
5
6  fb_image* backgroundImage;
7
8  int main(int argc, char *argv[]) {
```

```
 9      fb_init("/dev/graphics/fb0");
10      backgroundImage = fb_read_jpeg_image("/background.jpg");
11      clear_screen();
12
13      touch_init("/dev/input/event3");
14      int type, x, y, finger, i;
15      int lastX[5] = {0,0,0,0,0}, lastY[5] = {0,0,0,0,0};
16      int lastPressX = 0, lastPressY = 0, lastPressFinger = 0;
17      while (1) {
18          type = touch_read(&x, &y, &finger);
19
20          switch (type) {
21              case TOUCH_PRESS:
22                  printf("TOUCH_PRESS: x=%d,y=%d,finger=%d\n", x, y, finger);
23                  lastPressX = x, lastPressY = y, lastPressFinger = finger;
24                  break;
25              case TOUCH_MOVE: {
26                  int color = 0;
27                  int radius = 0;
28                  switch (finger) {
29                      case 0:
30                          radius = 17;
31                          color = 0xf13aa2;
32                          break;
33                      case 1:
34                          radius = 14;
35                          color = 0xad41e8;
36                          break;
37                      case 2:
38                          radius = 11;
39                          color = 0x6d02a8;
40                          break;
41                      case 3:
42                          radius = 8;
43                          color = 0xff3d3d;
44                          break;
45                      case 4:
46                          radius = 05;
47                          color = 0xff3d3d;
48                          break;
49                  }
50                  fb_draw_thick_line(lastX[finger], lastY[finger], x, y, radius,
                        color);
51                  fb_update();
52
53                  printf("TOUCH_MOVE: x=%d,y=%d,finger=%d\n", x, y, finger);
```

```
54                  break;
55              }
56              case TOUCH_RELEASE:
57                  printf("TOUCH_RELEASE: x=%d,y=%d,finger=%d\n", x, y, finger);
58                  if(lastPressX > 974 && lastPressY > 550 && lastPressFinger ==
                        finger &&
59                          x > 974 && y > 550){
60                      clear_screen();
61                  }
62                  break;
63              default:
64                  printf("unknown\n");
65                  break;
66          }
67          lastX[finger] = x, lastY[finger] = y;
68      }
69      return 0;
70  }
71
72  void clear_screen(){
73      fb_draw_image(0, 0, backgroundImage, 0);
74      fb_draw_image(974, 550, fb_read_png_image("/button_50.png"), 0);
75      //fb_draw_rect(0,0,1024,600,0);
76      fb_update();
77  }
```

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 }
```