目 录

1 实验一 1

1.1 实验目的 1
1.2 实验内容 1
1.3 实验调试 1
1.3.1 实验设计 1
1.3.2 实验调试及心得 2

2 实验二 4

2.1 实验目的 4
2.2 实验内容 4
2.3 实验调试 4
2.3.1 实验设计 4
2.3.2 实验调试及心得 6

3 实验三 10

3.1 实验目的 10
3.2 实验内容 10
3.3 实验调试 10
3.3.1 实验设计 10
3.3.2 实验调试及心得 11

4 实验四 14

4.1 实验目的 14
4.2 实验内容 14
4.3 实验调试 14
4.3.1 实验设计 14
4.3.2 实验调试及心得 15

5 实验五 18

5.1 实验目的 18
5.2 实验内容 18
5.3 实验调试 18
5.3.1 实验设计 18
5.3.2 实验调试及心得 19

附录 22

1 实验一

1.1 实验目的

核心编译，系统烧录。将 linux 系统烧入开发板中，并且进行简单应用程序的开发，使得能在开发板上运行。

1.2 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

1.3 实验调试

1.3.1 实验设计

1. Linux kernel 目录结构如下：
   - arch：与体系结构相关的代码，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；
2. 然后使用 make menucongif 命令，图像化配置内核。
3. 在核心源码目录下执行 make zImage 命令

4. 配置 usb 设备，在/etc/udev/rules.d/下创建一个配置文件 51-android.rules，在用户根目录下编辑 adb_usb.ini 文件。若没有该文件，创建一个，写入主设备号，十六进制以 0x 开头
5. 连接设备，打开串口准备烧写

1.3.2 实验调试及心得

实验调试：

首先需要确定 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

或者直接运行程序： adb shell /data/local/lab1

下图为实验一的测试结果，发现可以正确运行。

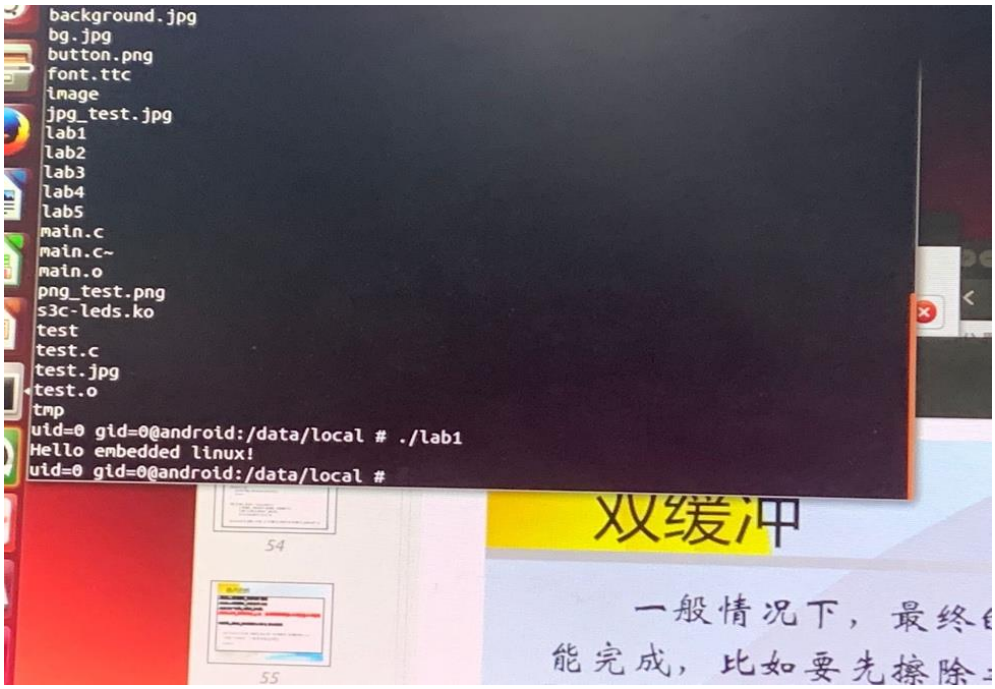


图 1.1 lab1 执行结果

实验心得：

在实验中遇到了几个问题。第一个就是板子与 PC 连接不上的问题，使用 lsusb 命令后发现没有 Google Inc 的信息，怎么重试也不行，后来发现是连接线坏了，导致接触不良。后来发生的问题是，将 linux 烧入后，启动板子，发现无论如何板子都不能进入 linux 界面，就是中间一条黄色的那个界面，后来看到 ppt 上面说的需要执行几条命令，才能恢复正常。

本次实验主要是一个实践性实验，将 ppt 上的步骤实现一边，通过本次实验，了解了系统烧录的方法，对嵌入式实验有一个初步的了解。

2 实验二

2.1 实验目的

Linux framebuffer 界面显示开发，能够显示基本的图形，比如点，先，矩形等。

2.2 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、矩形区域
3. 双缓冲机制

2.3 实验调试

2.3.1 实验设计

FrameBuffer 是出现在 2.2.xx 内核当中的一种驱动程序接口。

Linux 是工作在保护模式下，所以用户态进程是无法象 DOS 那样使用显卡 BIOS 里提供的中断调用来实现直接写屏，Linux 抽象出 FrameBuffer 这个设备来供用户态进程实现直接写屏。Framebuffer 机制模仿显卡的功能，将显卡硬件结构抽象掉，可以通过 Framebuffer 的读写直接对显存进行操作。用户可以将 Framebuffer 看成是显示内存的一个映像，将其映射到进程地址空间之后，就可以直接进行读写操作，而写操作可以立即反应在屏幕上。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。

但 Framebuffer 本身不具备任何运算数据的能力,就只好比是一个暂时存放水的水池.CPU 将运算后的结果放到这个水池,水池再将结果流到显示器. 中间不会对数据做处理. 应用程序也可以直接读写这个水池的内容.在这种机制下，尽管 Framebuffer 需要真正的显卡驱动的支持，但所有显示任务都有 CPU 完成,因此 CPU 负担很重。

基本原理

- 通过 framebuffer ，应用程序用 mmap 把显存映射到应用程序虚拟

地址空间，将要显示的数据写入这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_operations 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) ，LDC 控制器会自动的将显存中的数据显示在 LCD 屏上。

简单的讲，framebuffer 驱动的功能就是分配一块内存作显存，然后对 LCD 控制器的寄存器作一些设置。

具体来说：

- 填充一个 fbinfo 结构
- 用 reigster_framebuffer (fbinfo*) 将 fbinfo 结构注册到内核
- 对于 fbinfo 结构，最主要的是它的 fs_ops 成员，需要针对具体设备实现 fs_ops 中的接口
- 考虑是否使用中断处理
- 考虑内存访问方式
- 显卡不自带显存的，分配系统内存作为显存
- 显卡自带显存的，用 I/O 内存接口进行访问 (request_mem_region / ioremap)

双缓冲机制

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。

双缓冲绘图过程

1、所有的绘图函数都在后缓冲中绘图；

2、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)

3、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（framebuffer）； void fb_update(void);

4、清空需要更新的绘图区域；

双缓冲机制的扩展

1. 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

2. 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。

2.3.2 实验调试及心得

实验调试

画点函数已经给出，只需在画线函数和画矩形函数中调用画点函数实现对应对应功能即可。

1. fb_draw_line 的设计：

对坐标（x1，y1）与（x2，y2），分别计算 dx=x2-x1 与 dy=y2-y1，比较 dx 和 dy 的大小，选择跨度较大的轴作为之后 for 循环中具体绘图的方向。可以依据 x1，x2 的大小，决定 x 轴方向上绘图的方向，同理，可以依据 y1 和 y2 的大小，决定 y 轴上绘图的方向。对于每一个具体的像素点，调用 fb_draw_pixel 函数进行画点即可。

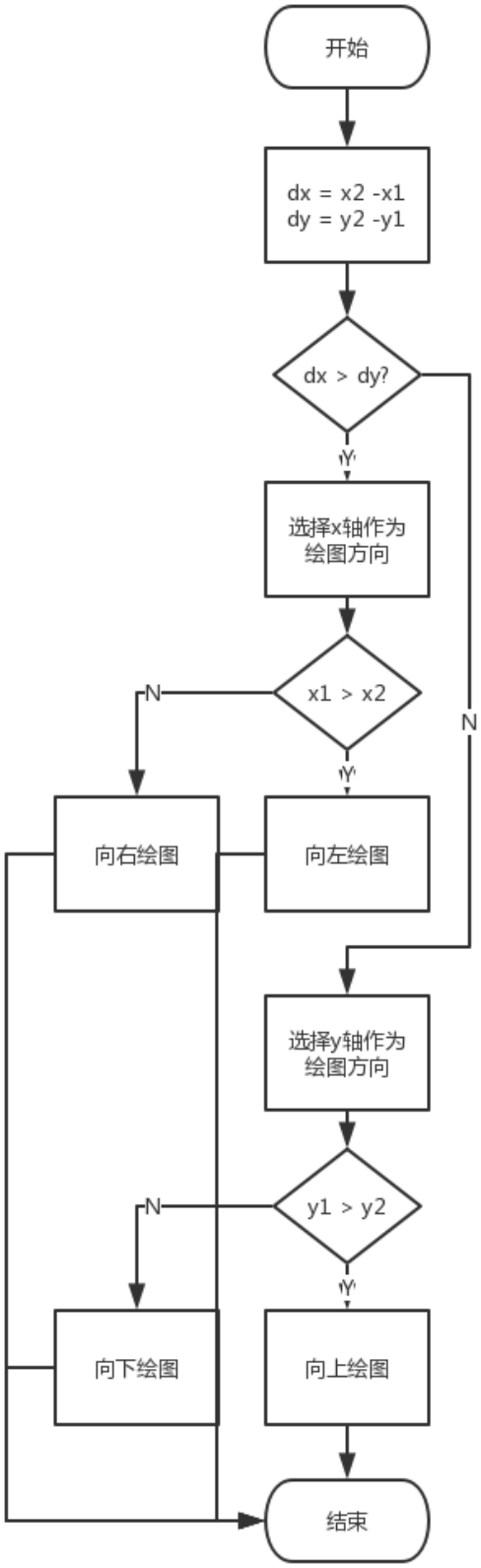

开始
dx = x2 -x1
dy = y2 -y1
dx > dy?
Y
N
选择x轴作为
绘图方向
x1 > x2
N
Y
向右绘图
向左绘图
选择y轴作为
绘图方向
y1 > y2
N
Y
向下绘图
向上绘图
结束

图 2.1 画线函数流程图

2. fb_draw_rect 的设计：

首先需要传入相应的参数确定矩形的四个顶点的坐标值，随后通过嵌套的 for 循环依次对矩形的每一列调用 fb_draw_pixel 函数绘制像素点即可。

测试结果如下图。

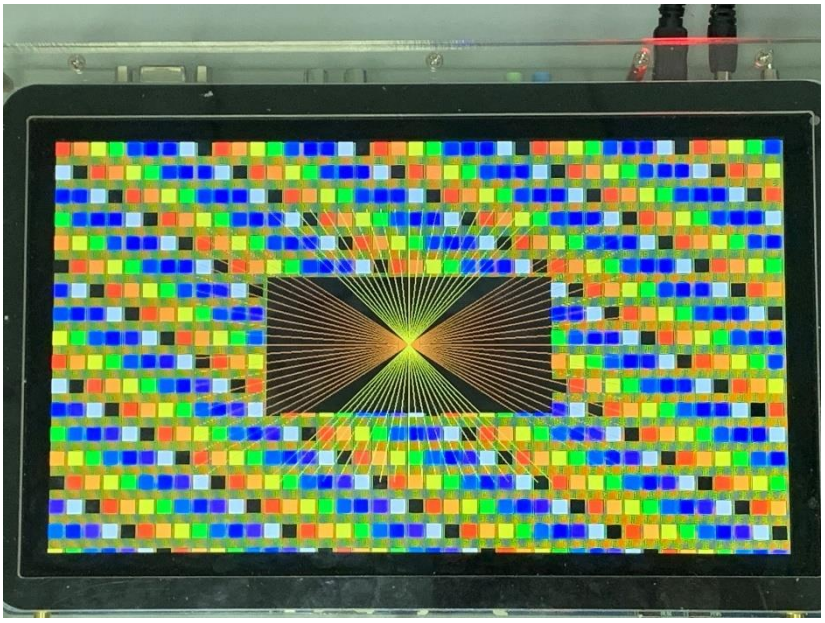


图 2.2 线、矩形绘制测试界面

线、矩形绘制的测试结果如下图所示。

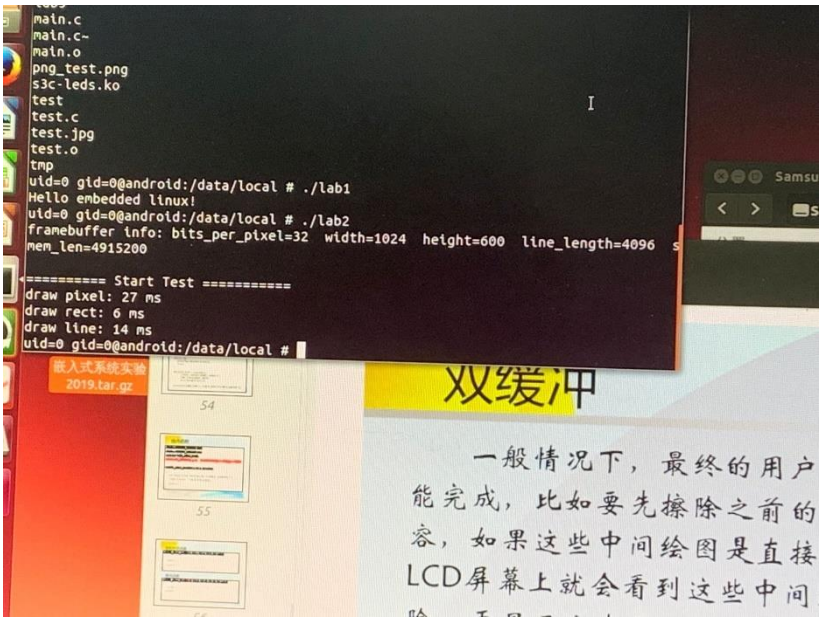


图 2.3 线、矩形绘制测试结果

实验心得

本次实验主要就是对实验机制的理解，理解内存拷贝函数的作用，需要将哪些参数传递进去。通过这次实验，能够在界面上画点，线，矩形，最后看到这些

图形，还是比较开心的。然后一个重要的评测标准就是绘图的速度，这个可以通过并发来加快程序速度。

3 实验三

3.1 实验目的

图片显示和文本显示，能够在界面上显示图片和文本。

3.2 实验内容

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

3.3 实验调试

3.3.1 实验设计

在实验二中已经了解了 framebuffer 机制，知道了如何在界面上显示点，线，矩形。这次就需要显示图片与文本。

实验中已经提供了图片的读取函数，能够给读取 jpeg 和 png 两种格式的图片，还能获取一个图片的子图片，最后需要释放图片。同时，实验也提供了文本的处理函数。所以我们只需要对这些函数的返回值进行处理即可。

在显示 PNG 图片和字体时需要支持 Alpha 通道以实现透明度的设置，alpha 有关知识在 ppt 中已经给的很详细了。

1. JPEG 图片显示

此图片格式不需要计算透明度，因此直接进行内存拷贝即可。

2. PNG 图片显示

PNG 图片涉及透明度，所以就需要便利所有的像素点，根据像素点的 Alpha 值以及对应 FrameBuffer 的三通道(R、G、B)值分别设置新的通道值并写入 FrameBuffer。如果 Alpha 值为 0 或者 255，就直接进行内存拷贝，不需要重新计算。

字体显示类似 PNG 显示的实现方式，对字体图像设置 Alpha 通道值、RGB 三通道值并进行输出即可，实验已经给出了字符串显示函数，因此在这里就不赘述了。

3.3.2实验调试及心得

实验调试

图片显示和文本显示测试界面如下图所示。

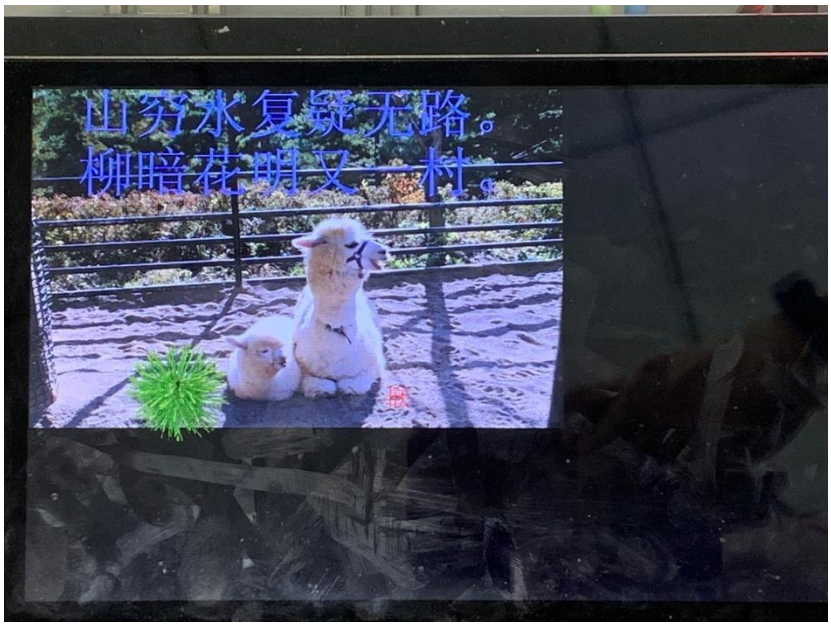


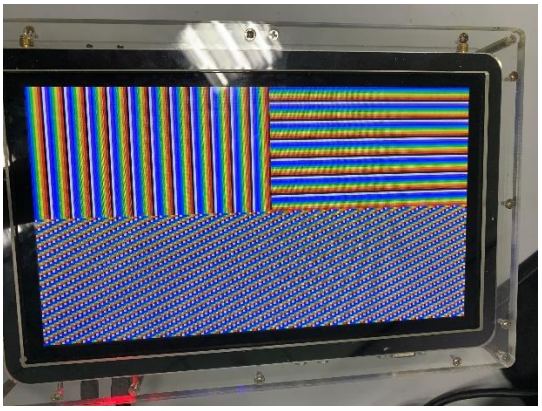
图 3.1 lab3 测试结果

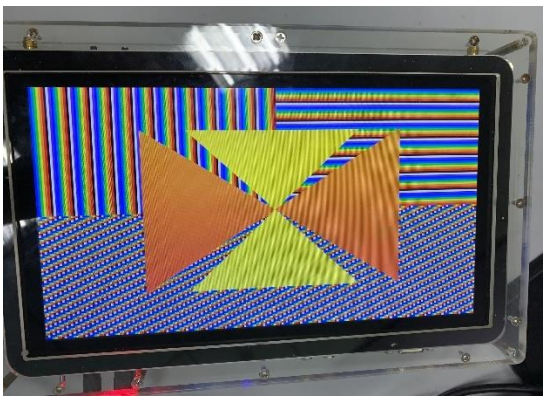
执行 test 函数。

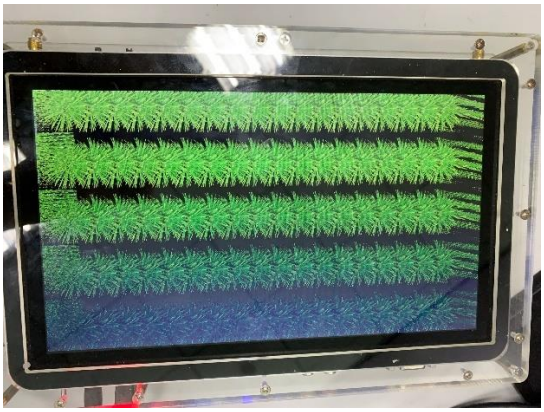
在测试进行前对绘制函数进行优化：为了在测试中提高绘制矩形的速度，在嵌套 for 循环的内层嵌套中，将每次绘制一列更改为一次绘制五列，从而提升了程序的运行速度。

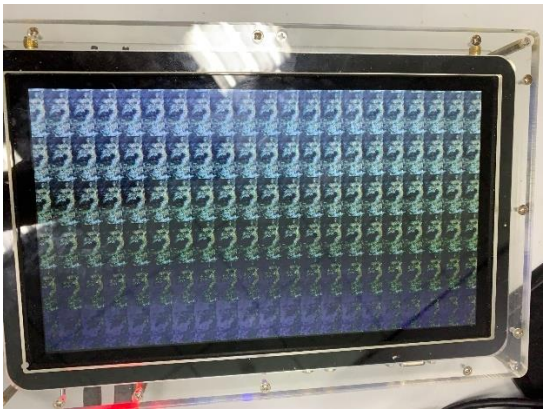
这样做的原理就是减少了循环次数，能够让 CPU 的效率更高。

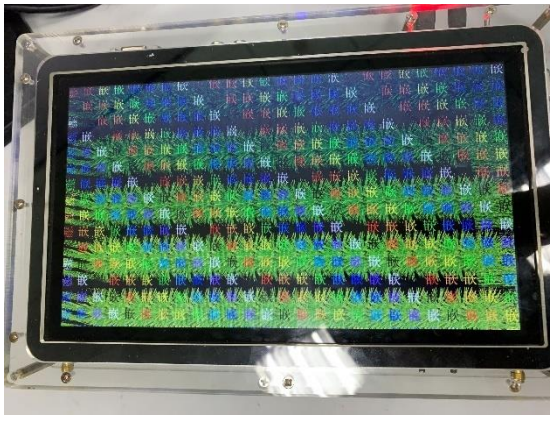
测试过程如图 3.2 所示











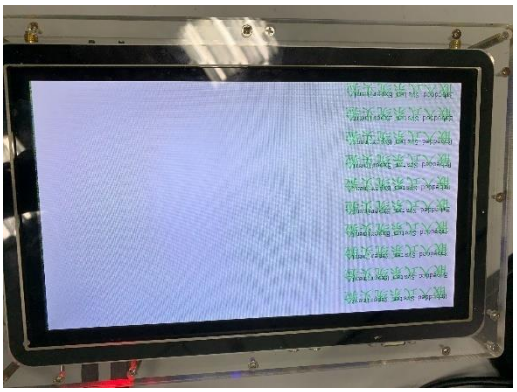


图 3.2 test 程序部分测试过程

test 程序测试结果如图 3.3 所示，程序测试结果 290ms。

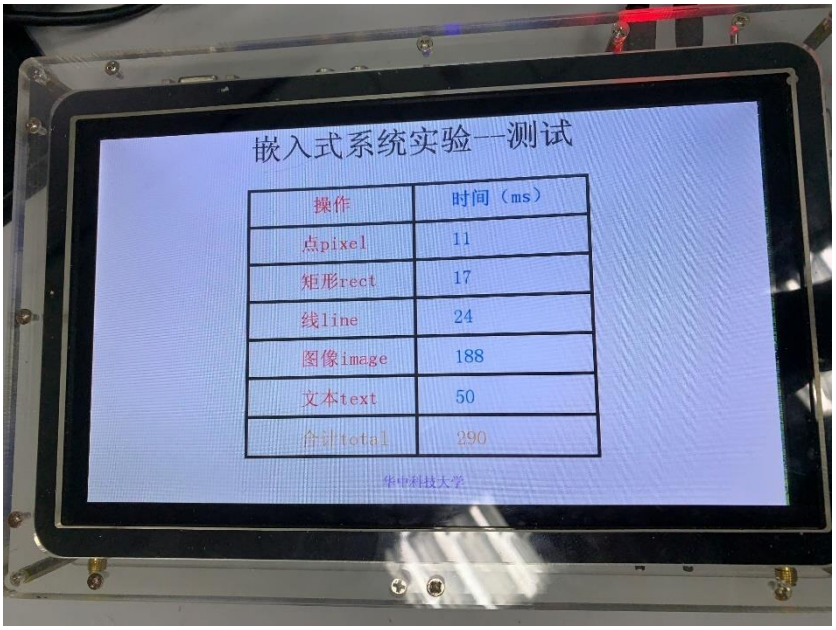


图 3.3test 程序部分测试结果

实验心得

本次实验内容和上次相似，但是难度加大。尤其是对各个操作的时间有限制，因此就需要对各个函数进行优化操作。本次优化的重点就是减少循环次数，能够提高效率，这样的话，就能理论上减少执行操作所需要的时间，最后验证也是如此。

4 实验四

4.1 实验目的

Linux touchscreen 多点触摸开发，能够实验触摸功能。

4.2 实验内容

1. Linux 下的触摸屏驱动接口
   - Input event 的使用
   - 多点触摸协议(Multi-touch Protocol)
2. 获取多点触摸的坐标

4.3 实验调试

4.3.1 实验设计

触摸屏的驱动接口已经给出。本实验采用多点触摸协议，即当一个触摸事件发生时，连续读取多条记录。只有当内容变化时，才发送。

本次实验中，触摸分为四类。分别是 TOUCH_NO_EVENT ，TOUCH_PRESS ，TOUCH_MOVE ， TOUCH_RELEASE 。分别将他们定义为 0，1，2，3.

同时，需要定义函数 touch_read 返回触摸信息，返回触摸的类型，坐标值，同时也返回 finger，即手指标记（0，1，2，3，4）。

需要在实验给定的 main 框架中实现自己选择的功能。

在这里我们选做手指触摸，要求：

1. 对应每个手指触点的圆的颜色不同；
2. 实时跟踪触点，只显示当前位置；
3. 之前位置的圆要清除掉；
4. 屏幕不能闪烁。

触摸类型有三种，按压，移动，释放。

1. 对于按压信号，需要在按压地区画圆，大小不需要太大，50×50 即可，同时五个圆需要呈现出不同的颜色。同时需要将圆心的坐标信息保存在数组中，命名为 old，便于需要将其删除时，可以读取数组中的信息，在指定的区域涂白（调用画圆函数，传入白色），这样就完成了消除的功能。为减小刷新全屏带来的延迟，减小屏幕闪烁的程度，在绘制完成后调用

fb_myupdate 函数仅仅重写该圆附近对应的内存，以将绘制结果输出在 LCD 屏幕上。

2. 对于移动信号，首先识别移动的是哪个圆，根据保存在数组中的信息，就可以获取这个圆之前坐标，需要将这个坐标的圆覆盖，即涂白。将新圆的圆心坐标信息覆盖先前的坐标，同时在屏幕的对应区域上绘制一个新的圆，除此之外，与 1 相似，为了减小屏幕的闪烁，就不能使用 fb_update 这种全局函数，需要使用自己的函数，即 fb_myupdate，这样只是刷新了一部份区域，减小内存拷贝带来的压力，闪烁程度能够有效减轻。
3. 对于释放信号，需要判断是哪个圆被释放了，判断完之后，需要从之前的 old 数组中获取那个圆原来的坐标信息，然后将那个位置刷新（对于本实验来说，背景是白色，因此只需要涂白即可），调用 fb_myupdate 函数。

4.3.2 实验调试及心得

实验调试

本次实验一开始做出来后，发现屏幕闪烁十分严重，调试也不行。后来寻求了老师的帮助，获得了两个思路。一个是帧同步的问题，另外一个就是全局刷新的问题。后来我们选择对全局刷新问题进行改善，每次刷新不需要对全屏刷新，而是只刷新特定区域，这样可以有效减小闪烁。

在测试过程中，发现屏幕闪烁程度可以得到较大的改善。但是，这种改善情况并不能完全消除闪烁，在快速移动手指的过程中，屏幕依然会出现按闪烁的情况。

测试结果如下图：

图 4.1 多点触碰测试

控制台输出如图 4.2 所示。

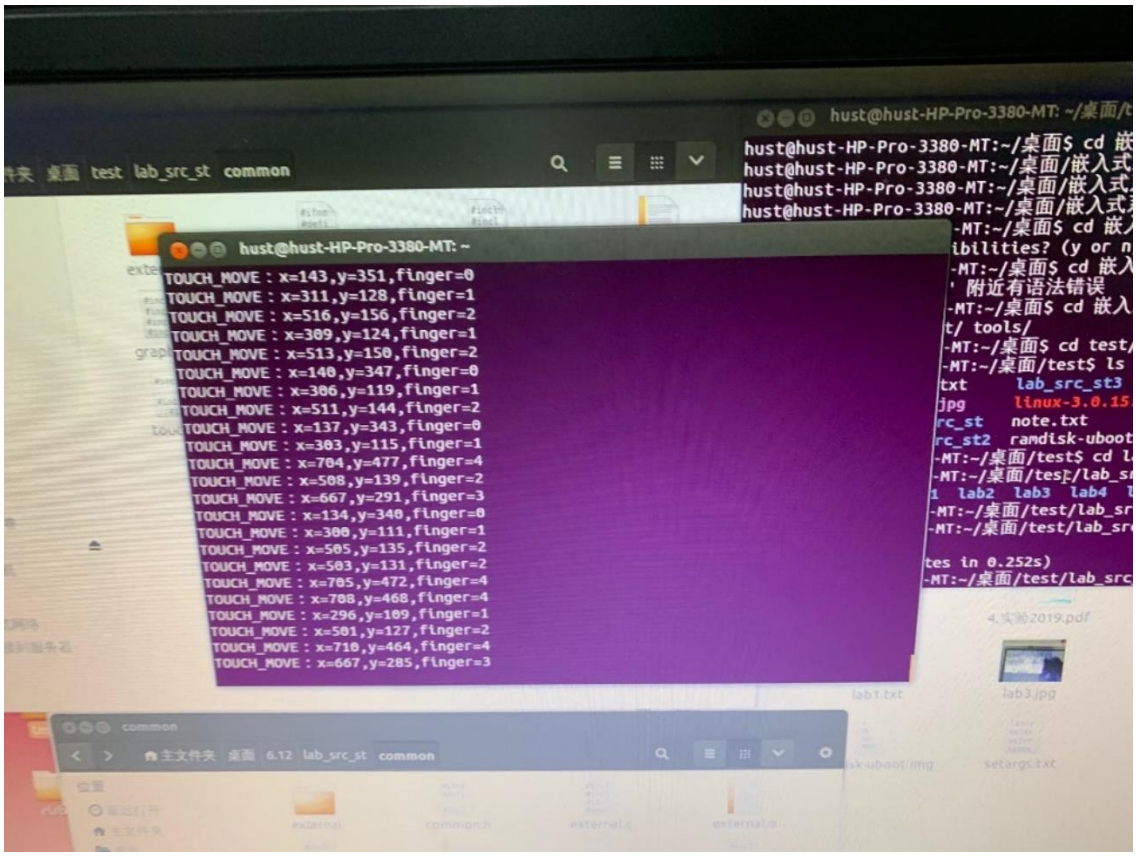


图 4.2 多点触碰控制台输出

实验心得

本次实验是第一次接触到触摸板开发，主要的过程已经在 ppt 中展示出来了，需要我们做的其实就是完成整个实验框架。对于这个实验，需要解决的主要就是屏幕闪烁的问题，这个问题到最后我们也没有解决完善，看起来只是调整刷新区域并不是最终的解决方案。如果需要进一步解决，可能需要使用多线程来对每一个刷新过程进行处理。或者查找资料，解决老师所说的帧同步的问题。

5 实验五

5.1 实验目的

Linux LED 驱动和控制界面。能够在屏幕上控制一个进度条，能够控制 LED 等的闪烁频率。

5.2 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

用拖动的方式来控制进度条的进度，在一边时，led 灯灭；在另一边时常亮；在中间时，led 灯闪烁，闪烁频率随位置调节。

5.3 实验调试

5.3.1 实验设计

LED 的闪烁不受到界面相关的影响，因此需要使用创建线程的方式完成这一独立执行的任务。

需要使用多线程来完成这些操作。绘制界面，将 LED 闪烁的频率写入一个变量保存。然后另外一个线程就需要读取这个频率值，然后控制 LED 灯。

实验中需要做出一个类似控制条的东西，设置 while 循环读取用户的操作，然后读取控制条上的百分比，这就与 LED 闪烁的频率相关，然后根据这个百分比设置 LED 闪烁的频率值。同时，还需要绘制背景，刷新进度条显示。

同时需要创建一个单独的线程来控制 LED 灯，读取 LED 灯的频率值，闪烁频率可以通过 usleep 函数操作 led 设备，也可以通过 led_set 函数操作 led 设备控制亮灭。

这样设置完成之后，整个程序就被分为了两个模块，一个主要用来进行界面的绘制，包括显示背景，显示进度条，刷新进度条。另外一个模块就用来设置 LED 相关。

这样做的好处就是能使两个线程之间互不干扰，防止一些可能会出现的问题。

5.3.2 实验调试及心得

实验调试

首先需要安装驱动模块

1．在主机用 adb push 命令将驱动模块上传至实验板

adb push s3c-leds.ko /data/local

2．用 adb shell 命令登陆实验板，执行安装模块命令 insmod /data/local/s3c-led.ko

3．登陆实验板 ，执行命令

cat /proc/devices 查看 led 驱动安装情况 查看 led 设备节点

实验能够通过进度条改变 LED 灯的闪烁频率，如下图

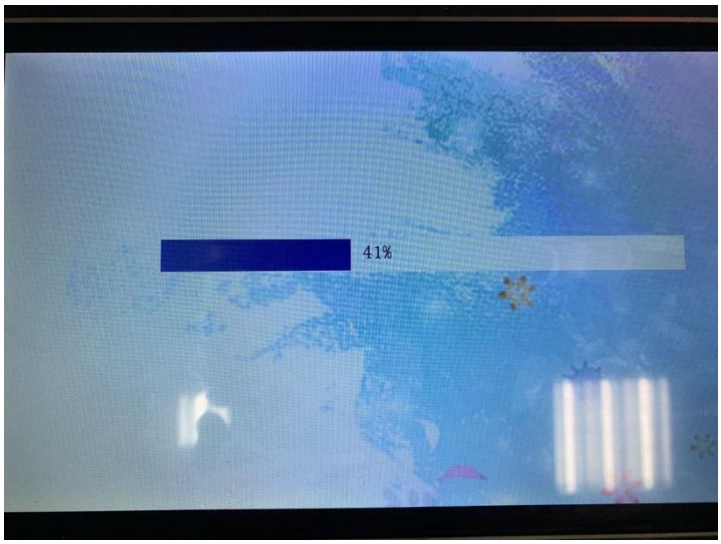


图 5.1 控制条

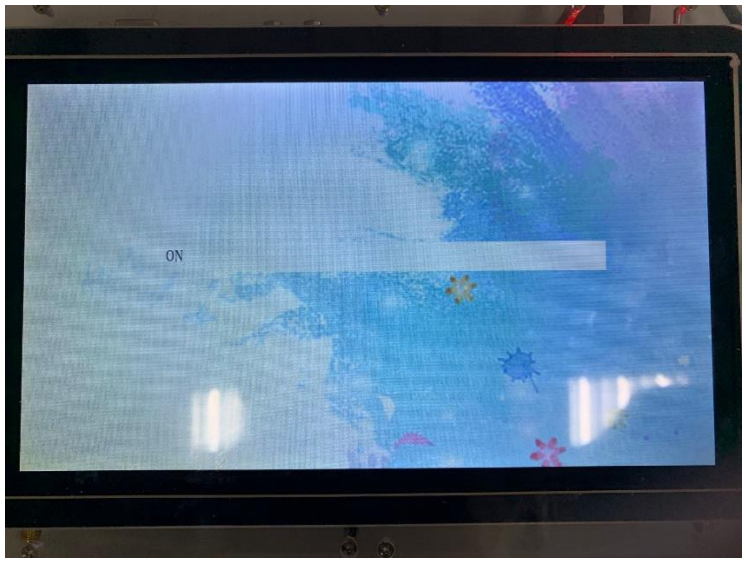


图 5.2 LED 灯常亮

图 5.3 LED 灯常亮

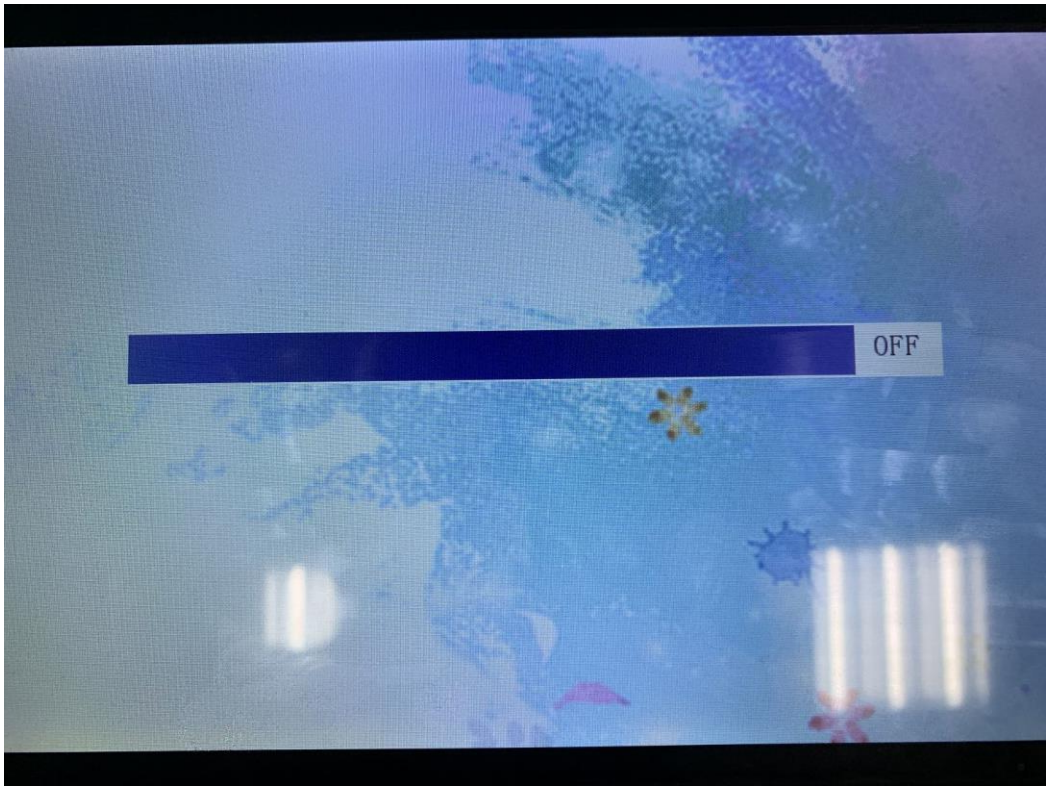


图 5.4 LED 灯熄灭

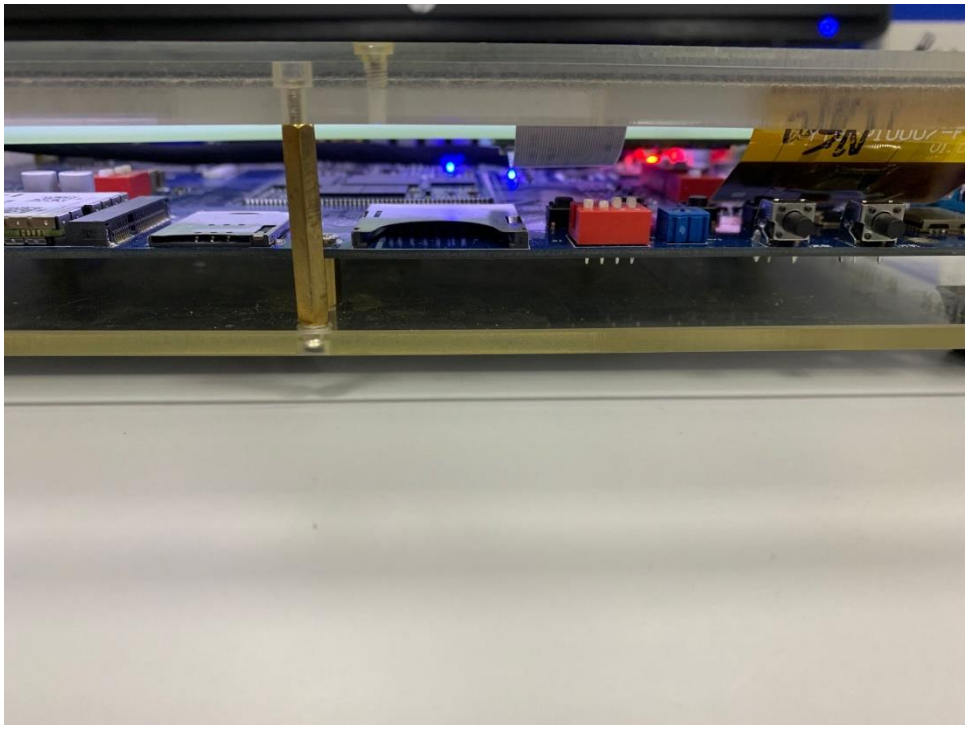


图 5.5 LED 灯熄灭

实验心得

本次实验主要就是控制 LED 灯，操作 led 设备。其中的界面设计在之前的实验中已经完成过相似的了，所以本次实验主要就是能够控制 LED 的亮灭以及闪烁频率。同时，根据 PPT 中的提示，需要新创建一个线程来单独控制 LED，这样可以防止一些意想不到的错误发生。他们之前的联系就是一个频率值，界面控制线程需要写频率值，而控制 led 线程需要读取这个值。

附录

```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;

static struct {
    int x1, y1, x2, y2;
} update_area = {0,0,0,0};


void fb_init(char *dev)
{
    int fd;
    struct fb_fix_screeninfo fb_fix;
    struct fb_var_screeninfo fb_var;

    if(LCD_FRAME_BUFFER != NULL) return; /*already done*/

    //First: Open the device
    if((fd = open(dev, O_RDWR)) < 0){
        printf("Unable to open framebuffer %s, errno = %d\n", dev, errno);
        return;
    }
    if(ioctl(fd, FBIOGET_FSCREENINFO, &fb_fix) < 0){
        printf("Unable to FBIOGET_FSCREENINFO %s\n", dev);
        return;
    }
    if(ioctl(fd, FBIOGET_VSCREENINFO, &fb_var) < 0){
        printf("Unable to FBIOGET_VSCREENINFO %s\n", dev);
        return;
    }

```

```
41.     printf("framebuffer info: bits_per_pixel=%u  width=%u  height=%u  line_l
    ength=%u  smem_len=%u\n",
42.             fb_var.bits_per_pixel, fb_var.xres, fb_var.yres, fb_fix.line_len
    gth, fb_fix.smem_len);
43.
44.     //Second: mmap
45.     int *addr;
46.     size_t size = fb_var.xres * fb_var.yres * fb_var.bits_per_pixel/8;
47.     addr = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
48.     if((int)addr == -1){
49.         printf("failed to mmap memory for framebuffer.\n");
50.         return;
51.     }
52.     LCD_FRAME_BUFFER = addr;
53.     return;
54. }
55.
56. /** copy data from mem buffer to frame buffer */
57. void fb_update(void)
58. {
59.     if(LCD_FRAME_BUFFER == NULL){
60.         printf("error: not allocate space for frame buffer\n");
61.         return;
62.     }
63.
64.     if((update_area.x1 >= SCREEN_WIDTH)
65.             || (update_area.x2 <= 0)
66.             || (update_area.y1 >= SCREEN_HEIGHT)
67.             || (update_area.y2 <= 0)) return;
68.
69.     int x,y,w,h;
70.     x = (update_area.x1 < 0) ? 0 : update_area.x1;
71.     y = (update_area.y1 < 0) ? 0 : update_area.y1;
72.     w = (update_area.x2 > SCREEN_WIDTH)?
73.         SCREEN_WIDTH - x : update_area.x2 - x;
74.     h = (update_area.y2 > SCREEN_HEIGHT)?
75.         SCREEN_HEIGHT - y : update_area.y2 - y;
76.
77.     int *src, *dst;
78.     src = LCD_MEM_BUFFER + y*SCREEN_WIDTH + x;
79.     dst = LCD_FRAME_BUFFER + y*SCREEN_WIDTH + x;
80.     while(h-- > 0){
81.         memcpy(dst, src, w*4);
82.         src += SCREEN_WIDTH;
```

```
83.         dst += SCREEN_WIDTH;
84.     }
85.
86.     update_area.x2 = 0;
87.     return;
88. }
89.
90. void fb_myupdate(int ux, int uy, int uw, int uh)
91. {
92.     ux-=50;
93.     uy-=50;
94.     uw = ux+uw;
95.     uh = uy+uh;
96.
97.
98.     if(LCD_FRAME_BUFFER == NULL){
99.         printf("error: not allocate space for frame buffer\n");
100.         return;
101.     }
102.
103.     if((ux >= SCREEN_WIDTH)
104.             || (uw <= 0)
105.             || (uy >= SCREEN_HEIGHT)
106.             || (uh <= 0)) return;
107.
108.     int x,y,w,h;
109.     x = (ux < 0) ? 0 : ux;
110.     y = (uy < 0) ? 0 : uy;
111.     w = (uw > SCREEN_WIDTH)?
112.         SCREEN_WIDTH - x : uw - x;
113.     h = (uh > SCREEN_HEIGHT)?
114.         SCREEN_HEIGHT - y : uh - y;
115.
116.
117.
118.     int *src, *dst;
119.     src = LCD_MEM_BUFFER + y*SCREEN_WIDTH + x;
120.     dst = LCD_FRAME_BUFFER + y*SCREEN_WIDTH + x;
121.     while(h-- > 0){
122.
123.         memcpy(dst, src, w*4);
124.         src += SCREEN_WIDTH;
125.         dst += SCREEN_WIDTH;
126.
```

```
127.
128.     }
129.
130.     ux = 0;
131.     return;
132. }
133.
134. void fb_myupdate2(int ux, int uy, int uw, int uh, int ox, int oy)
135. {
136.     ux-=50;
137.     uy-=50;
138.     uw = ux+uw;
139.     uh = uy+uh;
140.
141.     int ow;
142.     int oh;
143.     ox -= 50;
144.     oy -= 50;
145.     ow = ox+uw;
146.     oh = oy+uh;
147.
148.     if(LCD_FRAME_BUFFER == NULL){
149.         printf("error: not allocate space for frame buffer\n");
150.         return;
151.     }
152.
153.     if((ux >= SCREEN_WIDTH)
154.             || (uw <= 0)
155.             || (uy >= SCREEN_HEIGHT)
156.             || (uh <= 0)) return;
157.     if((ox >= SCREEN_WIDTH)
158.             || (ow <= 0)
159.             || (oy >= SCREEN_HEIGHT)
160.             || (oh <= 0)) return;
161.
162.     int x,y,w,h;
163.     x = (ux < 0) ? 0 : ux;
164.     y = (uy < 0) ? 0 : uy;
165.     w = (uw > SCREEN_WIDTH)?
166.         SCREEN_WIDTH - x : uw - x;
167.     h = (uh > SCREEN_HEIGHT)?
168.         SCREEN_HEIGHT - y : uh - y;
169.
170.     int oox,ooy,oow,ooh;
```

```
171.     oox = (ox < 0) ? 0 : ox;
172.     ooy = (oy < 0) ? 0 : oy;
173.     oow = (ow > SCREEN_WIDTH)?
174.         SCREEN_WIDTH - oox : ow - oox;
175.     ooh = (oh > SCREEN_HEIGHT)?
176.         SCREEN_HEIGHT - ooy : oh - ooy;
177.     int *osrc, *odst;
178.     osrc = LCD_MEM_BUFFER + ooy*SCREEN_WIDTH + oox;
179.     odst = LCD_FRAME_BUFFER + ooy*SCREEN_WIDTH + oox;
180. 
181.     int tempy = y<ooy?y:ooy;
182.     int tempx = x<oox?x:oox;
183.     int temph = h<ooh?ooh:h;
184.     int tempw = w<oow?oow:w;
185. 
186.     int *src, *dst;
187.     src = LCD_MEM_BUFFER + tempy*SCREEN_WIDTH + tempx;
188.     dst = LCD_FRAME_BUFFER + tempy*SCREEN_WIDTH + tempx;
189.     while(temph-- > 0){
190. 
191.         memcpy(dst, src, tempw*4);
192.         src += SCREEN_WIDTH;
193.         dst += SCREEN_WIDTH;
194. 
195.         //memcpy(odst, osrc, ow*4);
196.         //osrc += SCREEN_WIDTH;
197.         //odst += SCREEN_WIDTH;
198.     }
199. 
200.     ux = 0;
201.     return;
202. }
203. 
204. static void _update_area(int x, int y, int w, int h)
205. {
206.     //if((w <= 0)||(h <= 0)) return; /* sure */
207.     int x2 = x+w;
208.     int y2 = y+h;
209.     if(update_area.x2 == 0) {
210.         update_area.x1 = x;
211.         update_area.y1 = y;
212.         update_area.x2 = x2;
213.         update_area.y2 = y2;
214.     } else {
```

```
215.         if(update_area.x1 > x) update_area.x1 = x;
216.         if(update_area.y1 > y) update_area.y1 = y;
217.         if(update_area.x2 < x2) update_area.x2 = x2;
218.         if(update_area.y2 < y2) update_area.y2 = y2;
219.     }
220.     return;
221. }
222. 
223. /*====================================================================*/
224. 
225. void fb_draw_pixel(int x, int y, int color)
226. {
227.     if(x<0 || y<0 || x>=SCREEN_WIDTH || y>=SCREEN_HEIGHT) return;
228.     _update_area(x,y,1,1);
229.     int *tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
230.     *tmp = color;
231.     return;
232. }
233. 
234. void fb_draw_rect(int x, int y, int w, int h, int color)
235. {
236.     if(x < 0) { w += x; x = 0;}
237.     if(x+w > SCREEN_WIDTH) { w = SCREEN_WIDTH-x;}
238.     if(y < 0) { h += y; y = 0;}
239.     if(y+h >SCREEN_HEIGHT) { h = SCREEN_HEIGHT-y;}
240.     if(w<=0 || h<=0) return;
241.     _update_area(x,y,w,h);
242.     /*-----------------------------------------------------*/
243.     //printf("you need implement fb_draw_rect()\n"); exit(0);
244.     int i, j;
245.     int *tmp;
246.     for(i = y; i < y + h; i++){
247.         if (x>=(x+w-5)){
248.             for(j = x; j < x + w; j++){
249.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j;
250.                 *tmp = color;
251.             }
252.         }
253.         else{
254.             for(j = x; j < x + w-5; j+=5){
255.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j;
256.                 *tmp = color;
257.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j+1;
```

```
258.                 *tmp = color;
259.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j+2;
260.                 *tmp = color;
261.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j+3;
262.                 *tmp = color;
263.                 tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * i + j+4;
264.                 *tmp = color;
265.             }
266.         }
267.     }
268.     /*-------------------------------------------------*/
269.     return;
270. }
271.
272. void fb_draw_circle(int x, int y, int r, int color)
273. {
274.     if (x - r < 0 || y - r < 0 || x + r > SCREEN_WIDTH || y + r > SCREEN_HE
    IGHT) return;
275.     _update_area(x - r, y - r, 2 * r , 2 * r);
276.     int i, j;
277.     for (i = y - r ; i <= y + r ; ++i) {
278.         for (j = x - r ; j < x + r ; ++j) {
279.             if ((i - y) * (i - y) + (j - x) * (j - x) <= r * r) {
280.                 fb_draw_pixel(j, i, color);
281.             }
282.         }
283.     }
284. }
285.
286. void fb_draw_line(int x1, int y1, int x2, int y2, int color)
287. {
288. /*-------------------------------------------------*/
289.     //printf("you need implement fb_draw_line()\n"); exit(0);
290.     int x, y;
291.     int dx, dy;
292.     int ddx, ddy;
293.     int *tmp;
294.     int xmin, ymin, w, h;
295.     if(x1 < x2) xmin = x1, w = x2 - x1;
296.     else xmin = x2, w = x1 - x2;
297.     if(y1 < y2) ymin = y1, h = y2 - y1;
298.     else ymin = y2, h = y1 - y2;
299.     _update_area(xmin, ymin, w, h);
300.     dx = x2 - x1;
```

```
301.     dy = y2 - y1;
302.     ddx = (x2 > x1) ? 1: -1;
303.     ddy = (y2 > y1) ? 1: -1;
304.     if(dx > dy){
305.         for(x = x1; x != x2; x += ddx){
306.             y = (int)(((double)dy * (x - x1)) / dx) + y1;
307.             tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
308.             *tmp = color;
309.         }
310.     }
311.     else{
312.         for(y = y1; y != y2; y += ddy){
313.             x = (int)(((double)dx * (y - y1)) / dy) + x1;
314.             tmp = LCD_MEM_BUFFER + SCREEN_WIDTH * y + x;
315.             *tmp = color;
316.         }
317.     }
318. /*------------------------------------------------------*/
319.     return;
320. }
321.
322. /*======================================================================*/
323.
324. void fb_draw_image(int x, int y, fb_image *image, int color)
325. {
326.     if(image == NULL) return;
327.
328.     int ix = 0; //image x
329.     int iy = 0; //image y
330.     int w = image->pixel_w; //draw width
331.     int h = image->pixel_h; //draw height
332.
333.     if(x<0) {w+=x; ix-=x; x=0;}
334.     if(y<0) {h+=y; iy-=y; y=0;}
335.
336.     if(x+w > SCREEN_WIDTH) {
337.         w = SCREEN_WIDTH - x;
338.     }
339.     if(y+h > SCREEN_HEIGHT) {
340.         h = SCREEN_HEIGHT - y;
341.     }
342.     if((w <= 0)||(h <= 0)) return;
343.
```

```
344.     _update_area(x,y,w,h);
345. 
346.     char *dst = (char *)(LCD_MEM_BUFFER + y*SCREEN_WIDTH + x);
347.     char *src = image->content + iy*image->line_byte + ix*4;
348. /*-------------------------------------------------------------------*/
349. 
350.     int alpha;
351.     int ww;
352.     int i, j;
353.     char r, g, b;
354.     char *dst1, *src1, *dst2, *src2;
355.     int h1 = h * 4, w1 = w * 4;
356. 
357.     if(image->color_type == FB_COLOR_RGB_8880) /*lab3: jpg*/
358.     {
359.         //printf("you need implement fb_draw_image() FB_COLOR_RGB_8880\n"); exit(0);
360.         //memcpy(dst, src, w * h * 4);
361. 
362.         for(i = 0; i < h1; i += 4){
363.             dst1 = dst + i * SCREEN_WIDTH;
364.             src1 = src + i * image->pixel_w;
365.             memcpy(dst1, src1, w1);
366.             /*for(j = 0; j < w1; j += 4){
367.                 *(int*)(dst1 + j) = *(int*)(src1 + j);
368.             }*/
369.         }
370.         /*for(i = 0; i < h; i++){
371.             for(j = 0; j < w; j++){
372.                 *(int*)(dst + i * SCREEN_WIDTH * 4 + j * 4) = *(int*)(src + i * w * 4 + j * 4);
373.             }
374.         }*/
375.         return;
376.     }
377. 
378.     if(image->color_type == FB_COLOR_RGBA_8888) /*lab3: png*/
379.     {
380.         //printf("you need implement fb_draw_image() FB_COLOR_RGBA_8888\n"); exit(0);
381.         for(i = 0; i < h1; i += 4){
382.             src1 = src + i * w;
383.             dst1 = dst + i * SCREEN_WIDTH;
384.             for(j = 0; j < w1; j += 4){
```

```
385.                 dst2 = dst1 + j;
386.                 src2 = src1 + j;
387.                 alpha = *(src2 + 3);
388.                 switch(alpha){
389.                 case 255:
390.                     *(int*)(dst2) = *(int*)(src2);
391.                     break;
392.                 default:
393.                     b = *(src2);
394.                     g = *(src2 + 1);
395.                     r = *(src2 + 2);
396.                     *(dst2) += (((b - *(dst2)) * alpha) >> 8);
397.                     *(dst2 + 1) += (((g - *(dst2 + 1)) * alpha) >> 8);
398.                     *(dst2 + 2) += (((r - *(dst2 + 2)) * alpha) >> 8);
399.                     //*(dst2 + 3) = 255;
400.                 }
401.             }
402.         }
403.
404.         return;
405.     }
406.
407.     if(image->color_type == FB_COLOR_ALPHA_8) /*lab3: font*/
408.     {
409.         //printf("you need implement fb_draw_image() FB_COLOR_ALPHA_8\n");
    exit(0);
410.         for(i = 0; i < h; i++){
411.             for(j = 0; j < w; j++){
412.                 alpha = *(src + i * w + j);
413.                 switch(alpha){
414.                 case 255:
415.                     *(int*)(dst + i * SCREEN_WIDTH * 4 + j * 4) = color;
416.                     break;
417.                 default:
418.                     b = color & 0xff;
419.                     g = (color >> 8) & 0xff;
420.                     r = (color >> 16) & 0xff;
421.                     *(dst + i * SCREEN_WIDTH * 4 + j * 4) += (((b - *(dst +
     i * SCREEN_WIDTH * 4 + j * 4)) * alpha) >> 8);
422.                     *(dst + i * SCREEN_WIDTH * 4 + j * 4 + 1) += (((g - *(d
    st + i * SCREEN_WIDTH * 4 + j * 4 + 1)) * alpha) >> 8);
423.                     *(dst + i * SCREEN_WIDTH * 4 + j * 4 + 2) += (((r - *(d
    st + i * SCREEN_WIDTH * 4 + j * 4 + 2)) * alpha) >> 8);
424.                     *(dst + i * SCREEN_WIDTH * 4 + j * 4 + 3) = 255;
```

```
                    break;
                }
            }
        }
        return;
    }
/*---------------------------------------------------------------------*/
    return;
}

/** draw a text string **/
void fb_draw_text(int x, int y, char *text, int font_size, int color)
{
    fb_image *img;
    fb_font_info info;
    int i=0;
    int len = strlen(text);
    while(i < len)
    {
        img = fb_read_font_image(text+i, font_size, &info);
        if(img == NULL) break;
        fb_draw_image(x+info.left, y-info.top, img, color);
        fb_free_image(img);

        x += info.advance_x;
        i += info.bytes;
    }
    return;
}
```

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("Hello embedded linux!\n");
    return 0;
}
```

```
#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. }
```

```
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. {
```

```
    fb_init("/dev/graphics/fb0");
    font_init("/data/local/font.ttc");

    fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
    fb_update();

    fb_image *img;
    img = fb_read_jpeg_image("/data/local/jpg_test.jpg");
    fb_draw_image(0,0,img,0);
    fb_update();
    fb_free_image(img);

    img = fb_read_png_image("/data/local/png_test.png");
    fb_draw_image(100,300,img,0);
    fb_update();
    fb_free_image(img);

    img = fb_read_font_image("嵌",30,NULL);
    fb_draw_image(400,350,img,RED);
    fb_update();
    fb_free_image(img);

    fb_draw_text(50,50,"床前明月光，疑是地上霜。",64,PURPLE);
    fb_draw_text(50,120,"举头望明月，低头思故乡。",64,PURPLE);
    fb_update();
    return 0;
}
```

```
#include <stdio.h>
#include "../common/common.h"

int main(int argc, char *argv[])
{
    fb_init("/dev/graphics/fb0");
    fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,FB_COLOR(255,255,255));
    fb_update();

    touch_init("/dev/input/event3");
    int type,x,y,finger,i;
    int x_old[5], y_old[5];
    while(1){
        type = touch_read(&x,&y,&finger);
```

```
15.         switch(type){
16.             case TOUCH_PRESS:
17.                 switch(finger){
18.                     case 0:
19.                     fb_draw_circle(x,y,50,FB_COLOR(255,190,200));break;
20.                     case 1:
21.                     fb_draw_circle(x,y,50,FB_COLOR(200,190,255));break;
22.                     case 2:
23.                     fb_draw_circle(x,y,50,FB_COLOR(135,206,250));break;
24.                     case 3:
25.                     fb_draw_circle(x,y,50,FB_COLOR(127,255,170));break;
26.                     case 4:
27.                     fb_draw_circle(x,y,50,FB_COLOR(255,255,0));break;
28.                     default:
29.                     break;
30.                 }
31.                 //fb_update();
32.                 fb_myupdate(x,y,100,100);
33.                 printf("TOUCH_PRESS: x=%d,y=%d,finger=%d\n",x,y,finger);
34.                 break;
35.
36.             case TOUCH_MOVE:
37.                 fb_draw_circle(x_old[finger],y_old[finger],50,FB_COLOR(255,2
    55,255));
38.                 //fb_update();
39.                 //fb_myupdate(x_old[finger],y_old[finger],100,100);
40.                 switch(finger){
41.                     case 0:
42.                     fb_draw_circle(x,y,50,FB_COLOR(255,190,200));break;
43.                     case 1:
44.                     fb_draw_circle(x,y,50,FB_COLOR(200,190,255));break;
45.                     case 2:
46.                     fb_draw_circle(x,y,50,FB_COLOR(135,206,250));break;
47.                     case 3:
48.                     fb_draw_circle(x,y,50,FB_COLOR(127,255,170));break;
49.                     case 4:
50.                     fb_draw_circle(x,y,50,FB_COLOR(255,255,0));break;
51.                     default:
52.                     break;
53.                 }
54.                 fb_myupdate2(x,y,200,200,x_old[finger],y_old[finger]);
55.                 //fb_update();
56.                 printf("TOUCH_MOVE: x=%d,y=%d,finger=%d\n",x,y,finger);
57.                 break;
```

```
58.             case TOUCH_RELEASE:
59.                 fb_draw_circle(x_old[finger],y_old[finger],50,FB_COLOR(255,2
    55,255));
60.                 fb_myupdate2(x_old[finger],y_old[finger],200,200,x_old[finge
    r],y_old[finger]);
61.                 //fb_update();
62.                 printf("TOUCH_RELEASE: x=%d,y=%d,finger=%d\n",x,y,finger);
63.                 break;
64.             default:
65.                 break;
66.         }
67.
68.         x_old[finger] = x;
69.         y_old[finger] = y;
70.
71.     }
72.     return 0;
73.
74. }
```

```
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. int is_on;
22. int p;
```

```
23. int p_colors[10] = {0x2a36b1, 0x3b50ce, 0x455ede, 0x4e6cef, 0x4b69ff, 0x4d73
    ff, 0x5677fc, 0x6889ff, 0x738ffe, 0x91a7ff};
24. int status;
25. void led_set(int on)
26. {
27.     if(led_fd == -1)
28.     {
29.         led_fd = open("/dev/led", O_RDWR);
30.         if(led_fd < 0){
31.                     printf("open /dev/led failed, errno = %d\n", errno);
32.             return;
33.             }
34.     }
35.     on? ioctl(led_fd,LED_ON) : ioctl(led_fd,LED_OFF);
36.     return;
37. }
38. struct fingers {
39.     int status;
40.     int x;
41.     int y;
42. };
43. int colors[] = { 0xff0000, 0xffff00, 0xffff, 0xff00, 0xff};
44. void my_line(int dx, int dy, int sx, int sy, int color)
45. {
46. /*-------------------------------------------------*/
47.     //printf("you need implement fb_draw_line()\n"); exit(0);
48. if(abs(dy-sy) < abs(dx-sx))
49.     {
50.         int x;
51.         double k = (double)(dy - sy) / (dx - sx);
52.
53.         for(x=0; x<=abs(sx-dx); x++)
54.         {
55.             if(sx<dx)
56.                             fb_draw_rect(sx+x - 3,(int)(sy+k*x+0.5) - 3,6,6,c
    olor);
57.             else
58.                             fb_draw_rect(dx+x - 3,(int)(dy+k*x+0.5) - 3,6,6,
    color);
59.
60.         }
61.     }
62.     else
63.     {
```

```
64.         int y;
65.         double k = (double)(dx - sx) / (dy - sy);
66.
67.         for(y=0; y<=abs(sy-dy); y++)
68.         {
69.             if(sy<dy)
70.
71.                                 fb_draw_rect((int)(sx+k*y+0.5) - 3,sy+y -
    3 ,6,6, color);
72.             else
73.
74.                                   fb_draw_rect((int)(dx+k*y+0.5) - 3,dy+y -
    3,6,6, color);
75.         }
76.     }
77.     return;
78. }
79. /*=================================================================*/
80. void my_thread(){
81.     while (1) {
82.         if (p > 97) {led_set(0);usleep(100);}
83.         else if (p < 3) {led_set(1);usleep(100);}
84.         else { usleep(p * 1000); is_on = !is_on; led_set(is_on); }
85.     }
86. }
87.
88. struct historyText{
89.     int x;
90.     int y;
91. } hisText;
92.
93. /*绘制圆*/
94. void fb_draw_circle(int x, int y, int r,int color){
95.     int tempX=0;
96.     for(tempX=x-r; tempX<x+r;tempX++){
97.         int dy = (sqrt(r*r-abs(tempX-x)*abs(tempX-x))+0.5);
98.         if(tempX <=0) continue;
99.         /*绘制竖着的部分*/
100.          int tempY=0;
101.          for(tempY=y-dy; tempY<y+dy; tempY++){
102.               if(tempY <= 0) continue;
103.               if(tempY>= SCREEN_HEIGHT-1) break;
104.               fb_draw_pixel(tempX, tempY, color);
105.          }
```

```
106.     }
107. }
108. 
109. int draw_process(int process){
110.     int i = 0;
111.     fb_draw_rect(204, 254, (process - i) * 6-8, 42 , p_colors[i / 10]);
112.     char * tips;
113.     char t[4] = "00%";
114.     if (p < 3 ) {
115.         tips = "ON";
116.     }else if (p > 97) {
117.         tips = "OFF";
118.     }else{
119.         tips = t;
120.         tips[0] = p / 10 + '0';
121.         tips[1] = p % 10 + '0';
122.     }
123.     fb_draw_rect(hisText.x, hisText.y, 50, 25, FB_COLOR(255,255,255));
124.     fb_draw_text(183 + p * 6,230, tips, 25 , FB_COLOR(0,0,0));
125.     hisText.x = 183+p*6;
126.     hisText.y = 230-25;
127. }
128. int main(int argc, char* argv[])
129. {
130.     is_on = 0;
131. 
132.     fb_init("/dev/graphics/fb0");
133.     font_init("/data/local/font.ttc");
134.     fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,FB_COLOR(255,255,255));
135. 
136.     // set background jpeg
137.     fb_image *img;
138.     img = fb_read_jpeg_image("/data/local/jpg_test.jpg");
139.     fb_draw_image(0,0,img,0);
140.     fb_update();
141.     fb_free_image(img);
142. 
143.     pthread_t tid;
144.     pthread_create(&tid, NULL, my_thread , NULL);
145.     fb_update();
146.     struct fingers f[5];
147.     memset(f,0, sizeof(struct fingers) * 5);
148.     touch_init("/dev/input/event3");
149.     int type,x,y,finger,i;
```

```
150.     int flag;
151.     p = 0;
152.     while(1){
153.         fb_draw_rect(200,250,600,50, 0x000);
154.         fb_draw_rect(202,252,596,46, 0xffffff);
155.         draw_process(p);
156.         fb_update();
157.             type = touch_read(&x,&y,&finger);
158.             switch(type){
159.                 case TOUCH_PRESS:
160.                     //printf("TOUCH_PRESS:
    x=%d,y=%d,finger=%d\n",x,y,finger);
161.                     if (x < 800 && x > 200 && y < 300 && y > 250) status =
    1;
162.                     break;
163.                 case TOUCH_MOVE:
164.                     //printf("TOUCH_MOVE:
    x=%d,y=%d,finger=%d\n",x,y,finger);
165.                     if (status)
166.                         if (x >= 800) p = 100;
167.                         else if (x <= 200) p = 0;
168.                         else p = (x - 200 ) / 6;
169.                     break;
170.                 case TOUCH_RELEASE:
171.                     //printf("TOUCH_RELEASE:
    x=%d,y=%d,finger=%d\n",x,y,finger);
172.                     if ( x < 100 && y < 50) {is_on = !is_on; led_set(is_on)
    ;}
173.                     status = 0;
174.                     break;
175.                 default:
176.                     break;
177.             }
178.             fb_update();
179.     }
180.     return 0;
181. }
```

```
1.  /* Lab1 ~ Lab3 测试程序 */
2.
3.  #include <stdio.h>
4.  #include <time.h>
```

```
5.  #include <string.h>
6.
7.  #include "../common/common.h"
8.
9.  #define RED     FB_COLOR(255,0,0)
10. #define ORANGE  FB_COLOR(255,165,0)
11. #define YELLOW  FB_COLOR(255,255,0)
12. #define GREEN   FB_COLOR(0,255,0)
13. #define CYAN    FB_COLOR(0,127,255)
14. #define BLUE    FB_COLOR(0,0,255)
15. #define PURPLE  FB_COLOR(139,0,255)
16. #define WHITE   FB_COLOR(255,255,255)
17. #define BLACK   FB_COLOR(0,0,0)
18.
19. int main(int argc, char *argv[])
20. {
21.     int row,column,i;
22.     fb_image *img1,*img2,*img3;
23.
24.     fb_init("/dev/graphics/fb0");
25.     font_init("/data/local/font.ttc");
26.
27.     int32_t start ,end, time_total = 0;
28.     int32_t time_pixel=0,time_rect=0,time_image=0,time_text=0,time_line=0;
29.     int32_t time[5] = {0,0,0,0,0};
30.     int color[9] = {RED,ORANGE,YELLOW,GREEN,CYAN,BLUE,PURPLE,WHITE,BLACK};
31.
32.     fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
33.     fb_update();
34.     printf("\n========== Start Test ==========\n");
35.
36. //Lab2 test
37.     sleep(1);
38.     printf("\nLab2 test:\n");
39.     start = fb_get_time();
40.     for(row=-5; row<605; row+=2) {
41.         for(column=-5; column<1029; column+=2) {
42.             fb_draw_pixel(column,row,color[0]);
43.         }
44.     }
45.     end = fb_get_time();
46.     fb_update();
47.     time_pixel = end - start;
48.     time[0] = time_pixel;
```

```
49.     printf("draw pixel: %d ms\n",time_pixel);
50.     time_total += time_pixel;
51.
52.     sleep(1);
53.     start = fb_get_time();
54.     for(row=-15,i=0; row<300; row+=5){
55.         for(column=-15; column<1029; column+=5){
56.             fb_draw_rect(column,row,10,10,color[i]);
57.             i = (i+1) % 9;
58.         }
59.     }
60.     for(row=300,i=0; row<605; row+=5){
61.         fb_draw_rect(0,row,500,10,color[i]);
62.         i = (i+1) % 9;
63.     }
64.     for(column=500,i=0; column<1029; column+=5){
65.         fb_draw_rect(column,300,10,300,color[i]);
66.         i = (i+1) % 9;
67.     }
68.     end = fb_get_time();
69.     fb_update();
70.     time_rect = end - start;
71.     time[1] = time_rect;
72.     printf("draw rect: %d ms\n",time_rect);
73.     time_total += time_rect;
74.
75.     sleep(1);
76.     start = fb_get_time();
77.     for(row=0;row<=400;row++){
78.         fb_draw_line(500-300,300-200+row,500+300,300+200-row,color[1]);
79.     }
80.     for(column=0;column<=400;column++){
81.         fb_draw_line(500-200+column,300-200,500+200-
    column,300+200,color[2]);
82.     }
83.     end = fb_get_time();
84.     fb_update();
85.     time_line = end - start;
86.     time[2] = time_line;
87.     printf("draw line: %d ms\n", time_line);
88.     time_total += time_line;
89.
90. //Lab3 test
91.     sleep(1);
```

```
92.     printf("\nLab3 test:\n");
93.     img1 = fb_read_jpeg_image("/data/local/jpg_test.jpg");
94.     img2 = fb_read_png_image("/data/local/png_test.png");
95.     img3 = fb_read_font_image("嵌",30,NULL);
96.     start = fb_get_time();
97.     for(row=-5; row<605; row+=100){
98.         for(column=-5; column<1029; column+=50){
99.             fb_draw_image(column,row,img1,0);
100.         }
101.     }
102.     end = fb_get_time();
103.     fb_update();
104.     printf("    **jpeg:\t%d\n", end-start);
105.     time_image += (end - start);
106.
107.     sleep(1);
108.     fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
109.     start = fb_get_time();
110.     for(row=-5; row<605; row+=128){
111.         for(column=-5; column<1029; column+=50){
112.             fb_draw_image(column,row,img2,0);
113.         }
114.     }
115.     end = fb_get_time();
116.     fb_update();
117.     printf("    **png:\t%d\n", end-start);
118.     time_image += (end - start);
119.
120.     sleep(1);
121.     start = fb_get_time();
122.     for(row=-5,i=0; row<605; row+=40){
123.         for(column=-5; column<1029; column+=40){
124.             fb_draw_image(column,row,img3,color[i]);
125.             i = (i+1) % 9;
126.         }
127.     }
128.     end = fb_get_time();
129.     fb_update();
130.     printf("    **font:\t%d\n", end-start);
131.     time_image += (end - start);
132.
133.     time[3] = time_image;
134.     printf("draw image: %d ms\n", time_image);
135.     time_total += time_image;
```

```
136.     fb_free_image(img1);
137.     fb_free_image(img2);
138.     fb_free_image(img3);
139.
140.     sleep(1);
141.     fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,WHITE);
142.     start = fb_get_time();
143.     for(row=-5;row<605;row+=60){
144.         fb_draw_text(20,row,"嵌入式系统实验",40,color[3]);
145.         fb_draw_text(20,row+40,"Embedded System Experiment",20,color[8]);
146.     }
147.     end = fb_get_time();
148.     fb_update();
149.     time_text = end - start;
150.     time[4] = time_text;
151.     printf("draw text: %d ms\n",time_text);
152.     time_total += time_text;
153.
154.     printf("\nTotal time: %d ms\n\n", time_total);
155.     printf("==================================\n");
156.
157. //print result
158.     sleep(1);
159.     char time_array[10];
160.     fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,WHITE);
161.     fb_draw_text(255,50,"嵌入式系统实验--测试",50,BLACK);
162.     for(row=100,i=0; i<8; i++){
163.         fb_draw_rect(250,row,524,5,BLACK);
164.         row += 61;
165.     }
166.     for(column=250,i=0; i<3; i++){
167.         fb_draw_rect(column,100,5,432,BLACK);
168.         column += 260;
169.     }
170.     fb_draw_text(355,140,"操作",30,RED);
171.     fb_draw_text(570,140,"时间（ms）",30,CYAN);
172.     fb_draw_text(460,575,"华中科技大学",20,PURPLE);
173.     fb_draw_text(335,203,"点 pixel",30,RED);
174.     fb_draw_text(335,265,"矩形 rect",30,RED);
175.     fb_draw_text(335,327,"线 line",30,RED);
176.     fb_draw_text(335,389,"图像 image",30,RED);
177.     fb_draw_text(335,451,"文本 text",30,RED);
178.     fb_draw_text(335,513,"合计 total",30,ORANGE);
179.
```

```
180.     char str[5];
181.     for(i=0; i<5; i++){
182.         memset(str,'\0',5);
183.         sprintf(str,"%d",time[i]);
184.         fb_draw_text(570,203+62*i,str,30,CYAN);
185.     }
186.     memset(str,'\0',5);
187.     sprintf(str,"%d",time_total);
188.     fb_draw_text(570,513,str,30,ORANGE);
189. 
190.     fb_update();
191.     return 0;
192. }
```