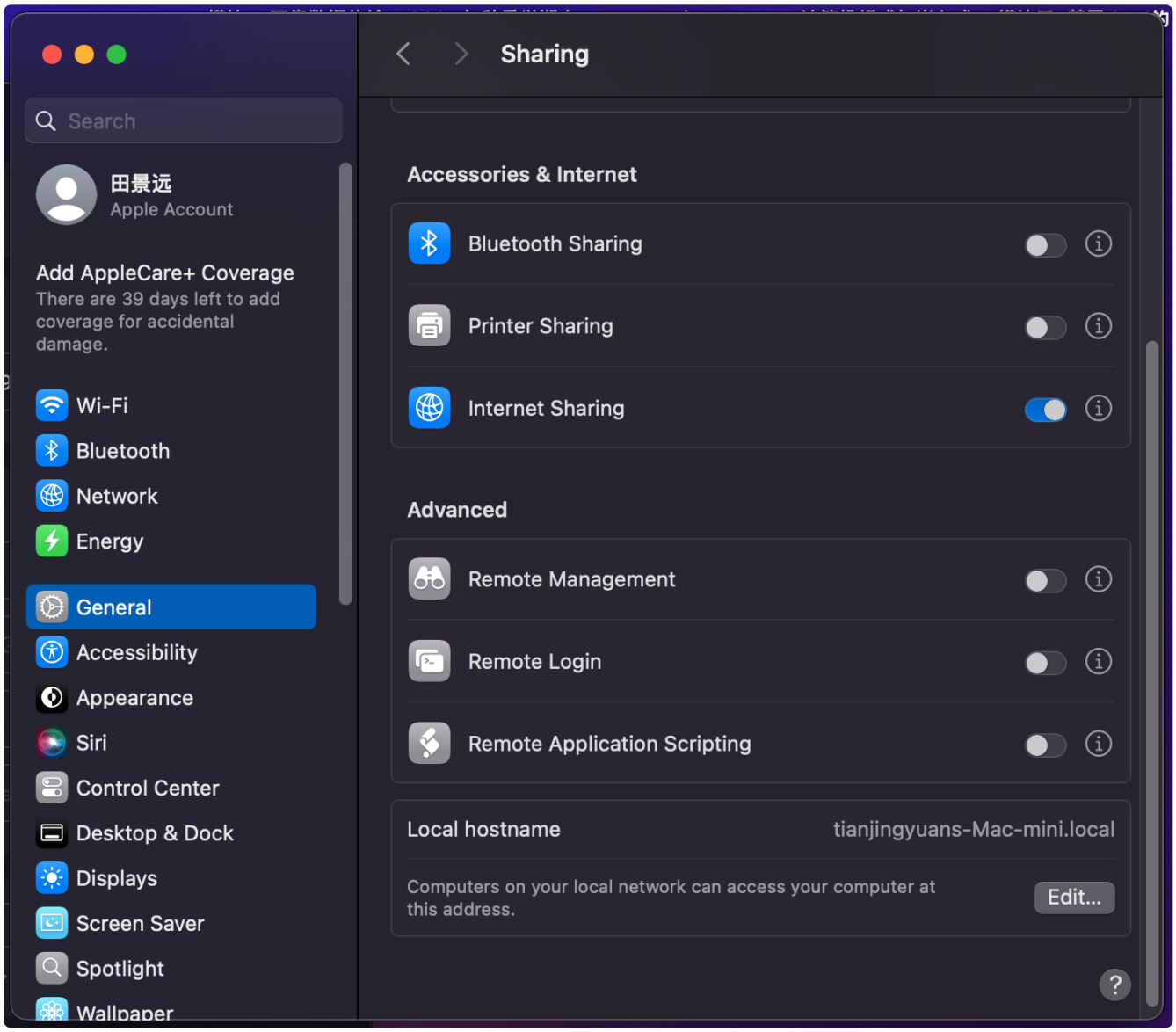
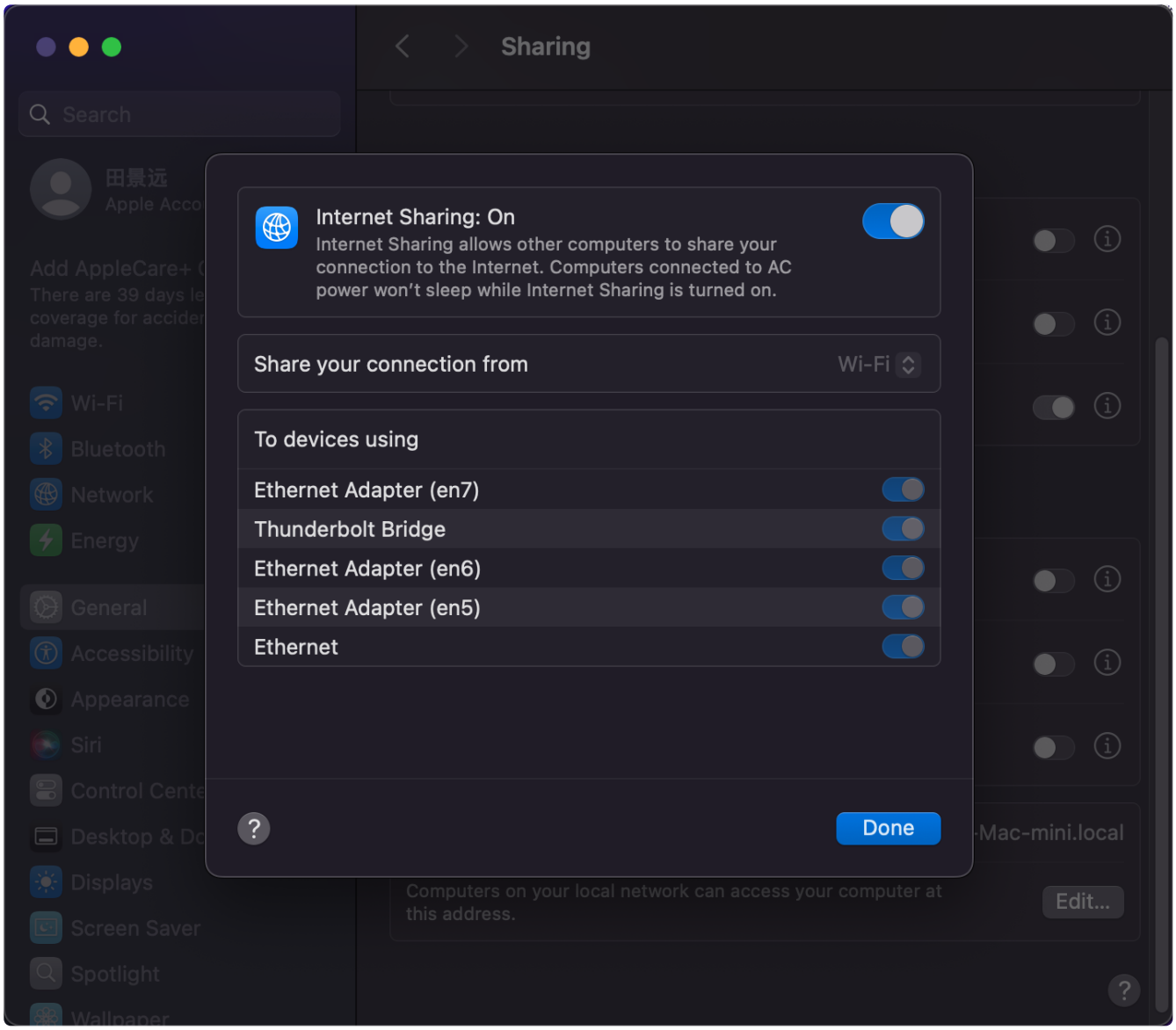
# 华中科技大学 嵌入式系统实验 macos 环境

## 开发板连接

我使用的是远程登录 ssh 的方式。

## 配置 macos 网络共享


Search
田景远
Apple Account
Add AppleCare+ Coverage
There are 39 days left to add coverage for accidental damage.
Wi-Fi
Bluetooth
Network
Energy
General
Accessibility
Appearance
Siri
Control Center
Desktop & Dock
Displays
Screen Saver
Spotlight
Wallpaper
Sharing
Accessories & Internet
Bluetooth Sharing
Printer Sharing
Internet Sharing
Advanced
Remote Management
Remote Login
Remote Application Scripting
Local hostname
tianjingyuans-Mac-mini.local
Computers on your local network can access your computer at this address.
Edit...

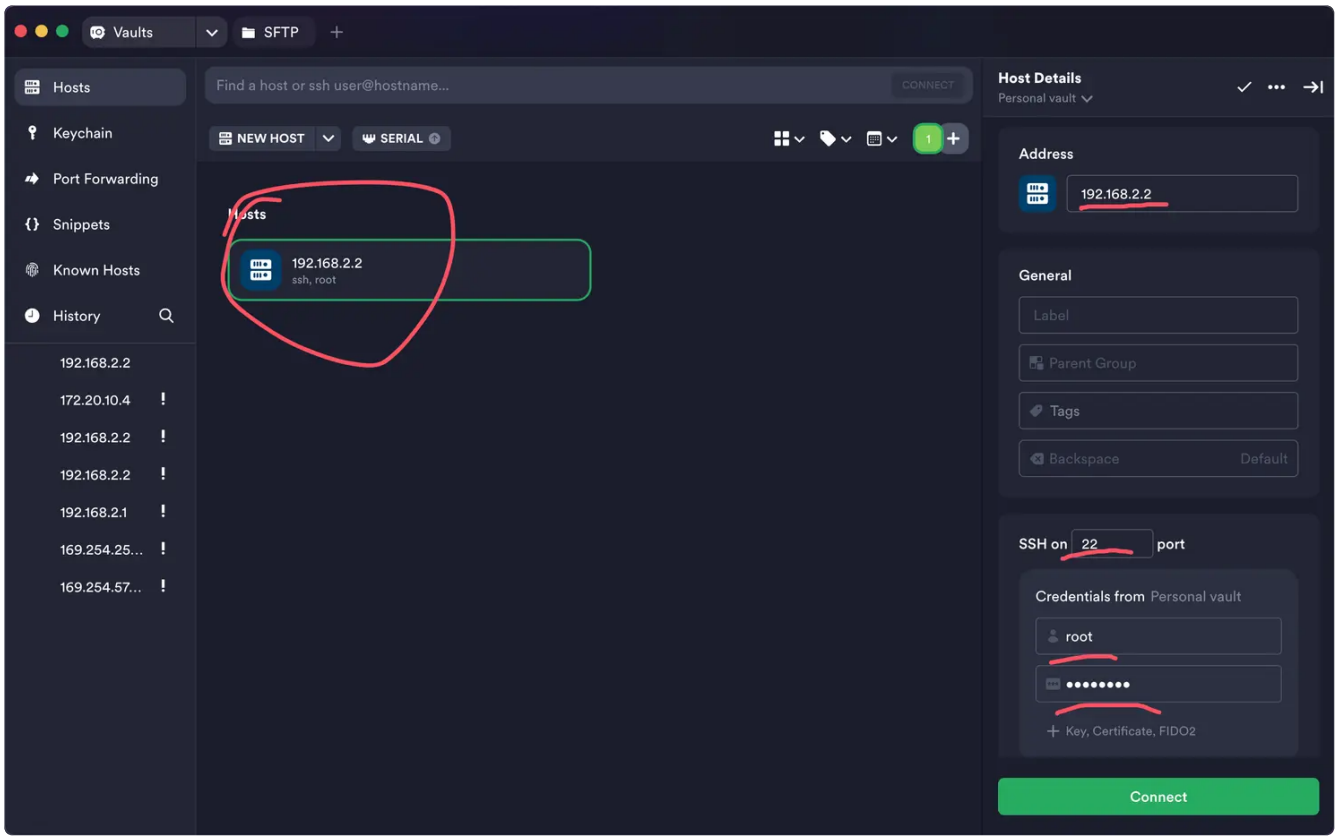


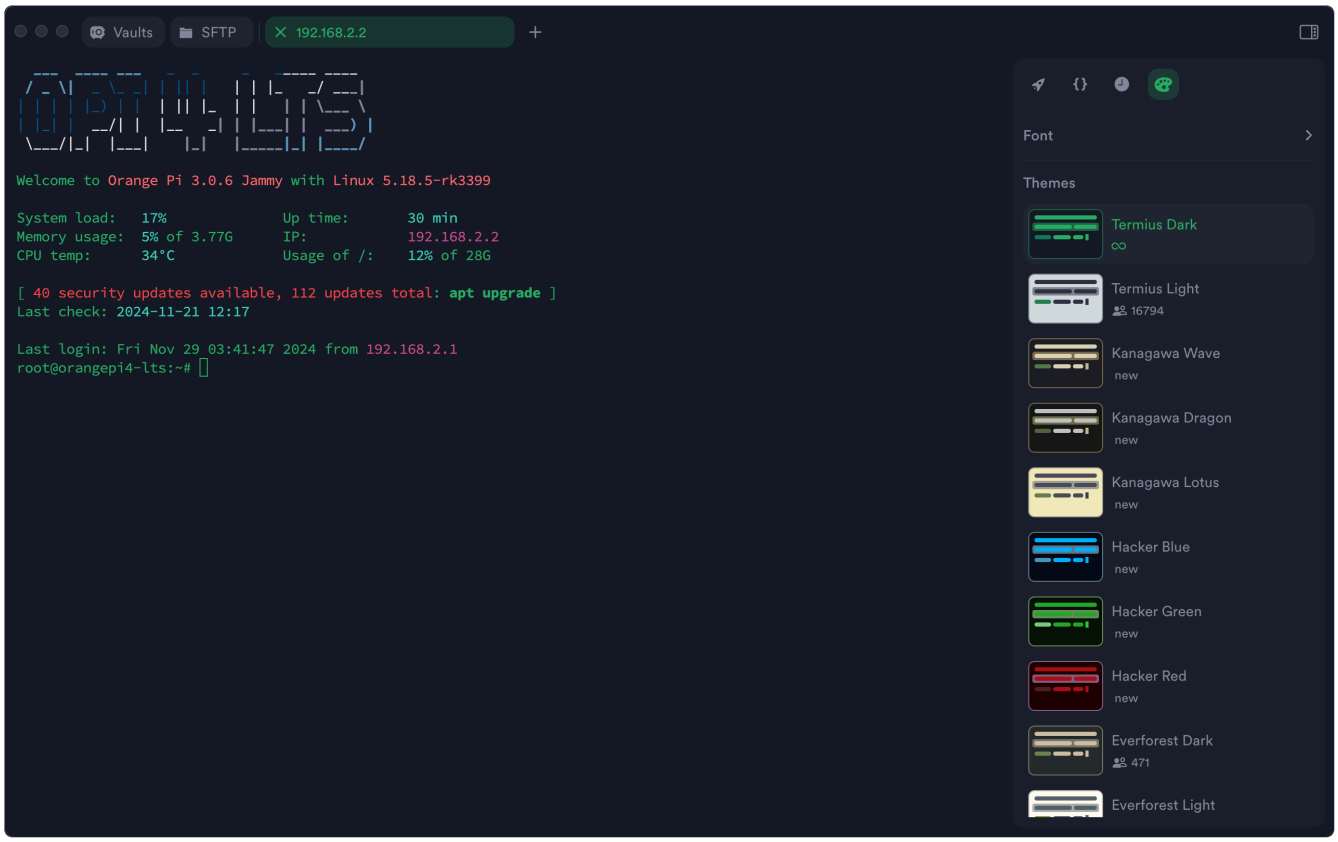

## 找到共享后主机的 ip 地址

```
flecthertian@tianjingyuans-Mac-mini ~ % arp -nla
Neighbor                Linklayer Address Expire(O) Expire(I)          Netif Refs Prbs
172.20.10.1             de:b5:4f:88:4:64  2m54s     2m21s           en1    1
172.20.10.4             e:89:3a:ca:89:db  (none)    (none)          en1
192.168.2.1             d2:11:e5:58:66:64 (none)    (none)      bridge10
224.0.0.251             1:0:5e:0:0:fb     (none)    (none)      bridge10
flecthertian@tianjingyuans-Mac-mini ~ % arp -nla
Neighbor                Linklayer Address Expire(O) Expire(I)          Netif Refs Prbs
172.20.10.1             de:b5:4f:88:4:64  2m52s     1m55s           en1    1
172.20.10.4             e:89:3a:ca:89:db  (none)    (none)          en1
192.168.2.1             d2:11:e5:58:66:64 (none)    (none)      bridge10
192.168.2.2             f6:cc:18:d6:96:61 1m18s     1m15s       bridge10    1
224.0.0.251             1:0:5e:0:0:fb     (none)    (none)      bridge10
```

在插入和拔出开发板的网线时，arp -nla 命令中新增 / 减少了 ip 地址 192.168.2.2，判断开发板的局域网地址为 192.168.2.2。

# SSH 登录


Vaults
SFTP
Hosts
Keychain
Port Forwarding
Snippets
Known Hosts
History
192.168.2.2
172.20.10.4
192.168.2.2
192.168.2.2
192.168.2.1
169.254.25...
169.254.57...
Find a host or ssh user@hostname...
CONNECT
NEW HOST
SERIAL
Hosts
192.168.2.2
ssh, root
Host Details
Personal vault
Address
192.168.2.2
General
Label
Parent Group
Tags
Backspace
Default
SSH on 22 port
Credentials from Personal vault
root
Key, Certificate, FIDO2
Connect


Vaults
SFTP
192.168.2.2
Welcome to Orange Pi 3.0.6 Jammy with Linux 5.18.5-rk3399
System load: 17%
Up time: 30 min
Memory usage: 5% of 3.77G
IP: 192.168.2.2
CPU temp: 34°C
Usage of /: 12% of 28G
[ 40 security updates available, 112 updates total: apt upgrade ]
Last check: 2024-11-21 12:17
Last login: Fri Nov 29 03:41:47 2024 from 192.168.2.1
root@orangepi4-lts:~#
Font
Themes
Termius Dark
Termius Light
16794
Kanagawa Wave
new
Kanagawa Dragon
new
Kanagawa Lotus
new
Hacker Blue
new
Hacker Green
new
Hacker Red
new
Everforest Dark
471
Everforest Light

登录成功。

## 远程传文件

如果说你的 terminus 无法正常拖放文件到本地，那么就需要下载官网的版本（app store 更新不及时）

我也把安装包放到了网盘

通过网盘分享的文件：Termius.dmg

链接：https://pan.baidu.com/s/1dHFd0yhSDjfxnjMx0r88iA?pwd=xfqb 提取码：xfqb

-- 来自百度网盘超级会员 v6 的分享

# 如何理解整个项目框架

首先在实验项目源目录下执行 make 指令，系统会执行 ./Makefile 中的内容。

每一个独立小实验都有一个自己的 makefile 文件，并且都依赖于一个公共的 rules.mk 构建方式。

## ./Makefile

Makefile

```
# 定义变量 DIRS，包含所有需要处理的目录
DIRS := lab1 lab2 lab3 test lab4 lab5 lab6

# 定义 all 目标，用于编译所有目录下的内容
all:
@for dir in $(DIRS); do make -C $$dir; done
# @表示不显示命令本身
# for 循环遍历 DIRS 中的每个目录
# make -C $$dir 表示进入目录 $$dir 执行 make

# 定义 clean 目标，用于清理所有目录下的生成文件
clean:
@for dir in $(DIRS); do make -C $$dir clean; done
# @表示不显示命令本身
# for 循环遍历 DIRS 中的每个目录
# make -C $$dir clean 表示进入目录 $$dir 执行 make clean
rm -f out/lab* out/test
# 删除 out 目录下以 lab 开头的文件和 test 文件
```

## 语法、原理和格式解释

**1. 变量定义：**

- `DIRS := lab1 lab2 lab3 test lab4 lab5 lab6`

- 使用 `:=` 进行变量赋值，表示递归展开右边的值。

- `DIRS` 包含所有需要处理的目录名称。

**2. ** 目标定义 **：**

- `all` 和 `clean` 是伪目标，不对应实际文件，用于执行特定操作。

**3. ** 命令执行 **：**

- 命令必须以 Tab 键开始，不能是空格。

- `@` 符号抑制命令回显，只显示命令输出。

- `$$dir` 在 Makefile 中表示一个 `$dir`，因为 Makefile 中 `$` 需要转义。

**4. ** 循环和条件 **：**

- 使用 shell 循环 `for dir in $(DIRS); do ...; done` 遍历目录列表。
- `make -C $$dir` 进入目录执行 make。
- `make -C $$dir clean` 进入目录执行 make clean。

**5. ** 文件删除 **：**

- `rm -f out/lab* out/test` 删除指定文件，`-f` 表示强制删除不提示。

## 原理和流程

**- **Makefile 执行流程 **：**

- 执行 `make` 时，默认执行第一个目标 `all`。
- 执行 `all` 目标时，循环进入每个目录执行 `make`。
- 每个目录下的 Makefile 负责编译该目录下的代码。
- 执行 `make clean` 时，循环进入每个目录执行 `make clean`，然后删除指定文件。

**- ** 变量替换 **：**

- 在 Makefile 中，变量名用 `$(VAR)` 或 `${VAR}` 表示，Make 会自动替换为变量的值。

**- ** 伪目标 **：**

- `all` 和 `clean` 是伪目标，不对应实际文件，用于执行某些操作。

## 格式和注意事项

- ** 缩进 **:

  - 命令必须以 Tab 键开始，不能是空格，否则会报错。

- ** 变量引用 **:

  - 在 Makefile 中，变量引用需要使用 `$`，并且需要转义，因此用 `$$` 表示一个 `$`。

- ** 命令执行顺序 **:

  - Makefile 中，目标的依赖会按顺序执行，命令也是按顺序执行的。

通过这些注释和解释，我们可以清晰地理解这段 Makefile 的功能和实现原理，以及其在编译和清理多个目录下的代码时的作用。

## ./common/rules.mk

Makefile

```
# 指定交叉编译工具链的路径前缀
CROSS_COMPILE := /home/oliver_cai/orangepi4-code/gcc-aarch64/bin/aarch64-none-linux-gnu-

# 设置编译器为交叉编译器
CC := $(CROSS_COMPILE)gcc

# 编译选项：启用所有警告，优化级别2
CFLAGS := -Wall -O2

# 链接选项：启用所有警告
LDFLAGS := -Wall

# 指定包含目录，用于查找头文件
INCLUDE := -I../common/external/include

# 指定库目录及要链接的库
LIB := -L../common/external/lib -ljpeg -lfreetype -lpng -lasound -lz -lc -lm

# 列出可执行文件的源文件
# 注意：这里原来EXESRCS被错误地追加了自身，应修正为实际的源文件列表
EXESRCS := ../common/graphic.c ../common/touch.c ../common/image.c ../common/task.c

# 将源文件列表中的 .c 文件转换为 .o 文件
EXEOBJS := $(patsubst %.c, %.o, $(EXESRCS))

# 构建可执行文件 $(EXENAME) 的规则
$(EXENAME): $(EXEOBJS)
    $(CC) $(LDFLAGS) -o $(EXENAME) $(EXEOBJS) $(LIB)
    mv $(EXENAME) ../out/

# 清理生成的可执行文件和对象文件
clean:
    rm -f $(EXENAME) $(EXEOBJS)

# 编译 .c 文件为 .o 文件的通用规则，依赖于 ../common/common.h
```

```
36 %.o: %.c ../common/common.h
37 	$(CC) $(CFLAGS) $(INCLUDE) -c -o $@ $<
```

# 实验一

进入 lab 根目录，命令行执行 make 就能在 out 文件夹中看到 lab1 的输出程序了，再 cd 到 out 文件夹中 ./lab1，即可输出 hello world

# 实验二、实验三

lab2 和 lab3 中的 main.c 代码都是可以直接编译执行的，学生需要补充 /common/graphic.c 中被 main

函数调用的绘图函数的代码。

## graphic.c

引用出处：https://github.com/Oliver-242/HUST-Embedded-System

C/C++

```
#include "common.h"
#include <sys/ioctl.h>
#include <linux/fb.h>
#include <linux/kd.h>
#include <linux/vt.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>

#define RED    FB_COLOR(255,0,0)
#define ORANGE    FB_COLOR(255,165,0)
#define YELLOW    FB_COLOR(255,255,0)
#define GREEN    FB_COLOR(0,255,0)
#define CYAN    FB_COLOR(0,127,255)
#define BLUE    FB_COLOR(0,0,255)
#define PURPLE    FB_COLOR(139,0,255)
#define WHITE   FB_COLOR(255,255,255)
#define BLACK   FB_COLOR(0,0,0)

static int LCD_FB_FD;
static int *LCD_FB_BUF = NULL;
static int DRAW_BUF[SCREEN_WIDTH*SCREEN_HEIGHT];

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

#define AREA_SET_EMPTY(pa) do {\
    (pa)->x1 = SCREEN_WIDTH;\
    (pa)->x2 = 0;\
    (pa)->y1 = SCREEN_HEIGHT;\
    (pa)->y2 = 0;\
} while(0)

```

```
39 void fb_init(char *dev)
40 {
41     int fd;
42     struct fb_fix_screeninfo fb_fix;
43     struct fb_var_screeninfo fb_var;
44 
45     if(LCD_FB_BUF != NULL) return; /*already done*/
46 
47     //进入终端图形模式
48     fd = open("/dev/tty0",O_RDWR,0);
49     ioctl(fd, KDSETMODE, KD_GRAPHICS);
50     close(fd);
51 
52     //First: Open the device
53     if((fd = open(dev, O_RDWR)) < 0){
54         printf("Unable to open framebuffer %s, errno = %d\n", dev, errn
   o);
55         return;
56     }
57     if(ioctl(fd, FBIOGET_FSCREENINFO, &fb_fix) < 0){
58         printf("Unable to FBIOGET_FSCREENINFO %s\n", dev);
59         return;
60     }
61     if(ioctl(fd, FBIOGET_VSCREENINFO, &fb_var) < 0){
62         printf("Unable to FBIOGET_VSCREENINFO %s\n", dev);
63         return;
64     }
65 
66     printf("framebuffer info: bits_per_pixel=%u,size=(%d,%d),virtual_po
   s_size=(%d,%d)(%d,%d),line_length=%u,smem_len=%u\n",
67         fb_var.bits_per_pixel, fb_var.xres, fb_var.yres, fb_var.xoffse
   t, fb_var.yoffset,
68         fb_var.xres_virtual, fb_var.yres_virtual, fb_fix.line_length, f
   b_fix.smem_len);
69 
70     //Second: mmap
71     void *addr = mmap(NULL, fb_fix.smem_len, PROT_READ|PROT_WRITE, MAP
   _SHARED, fd, 0);
72     if(addr == (void *)-1){
73         printf("failed to mmap memory for framebuffer.\n");
```

```
74          return;
75      }
76
77      if((fb_var.xoffset != 0) ||(fb_var.yoffset != 0))
78      {
79          fb_var.xoffset = 0;
80          fb_var.yoffset = 0;
81          if(ioctl(fd, FBIOPAN_DISPLAY, &fb_var) < 0) {
82              printf("FBIOPAN_DISPLAY framebuffer failed\n");
83          }
84      }
85
86      LCD_FB_FD = fd;
87      LCD_FB_BUF = addr;
88
89      //set empty
90      AREA_SET_EMPTY(&update_area);
91      return;
92  }
93
94  static void _copy_area(int *dst, int *src, struct area *pa)
95  {
96      int x, y, w, h;
97      x = pa->x1; w = pa->x2-x;
98      y = pa->y1; h = pa->y2-y;
99      src += y*SCREEN_WIDTH + x;
100     dst += y*SCREEN_WIDTH + x;
101     while(h-- > 0){
102         memcpy(dst, src, w*4);
103         src += SCREEN_WIDTH;
104         dst += SCREEN_WIDTH;
105     }
106 }
107
108 static int _check_area(struct area *pa)
109 {
110     if(pa->x2 == 0) return 0; //is empty
111
112     if(pa->x1 < 0) pa->x1 = 0;
113     if(pa->x2 > SCREEN_WIDTH) pa->x2 = SCREEN_WIDTH;
```

```
114     if(pa->y1 < 0) pa->y1 = 0;
115     if(pa->y2 > SCREEN_HEIGHT) pa->y2 = SCREEN_HEIGHT;
116 
117     if((pa->x2 > pa->x1) && (pa->y2 > pa->y1))
118         return 1; //no empty
119 
120     //set empty
121     AREA_SET_EMPTY(pa);
122     return 0;
123 }
124 
125 void fb_update(void)
126 {
127     if(_check_area(&update_area) == 0) return; //is empty
128     _copy_area(LCD_FB_BUF, DRAW_BUF, &update_area);
129     AREA_SET_EMPTY(&update_area); //set empty
130     return;
131 }
132 
133 /*=====================================================================
    =*/
134 
135 static void * _begin_draw(int x, int y, int w, int h)
136 {
137     int x2 = x+w;
138     int y2 = y+h;
139     if(update_area.x1 > x) update_area.x1 = x;
140     if(update_area.y1 > y) update_area.y1 = y;
141     if(update_area.x2 < x2) update_area.x2 = x2;
142     if(update_area.y2 < y2) update_area.y2 = y2;
143     return DRAW_BUF;
144 }
145 
146 void fb_draw_pixel(int x, int y, int color)
147 {
148     if(x<0 || y<0 || x>=SCREEN_WIDTH || y>=SCREEN_HEIGHT) return;
149     int *buf = _begin_draw(x,y,1,1);
150 /*-----------------------------------------------------*/
151     *(buf + y*SCREEN_WIDTH + x) = color;
152 /*-----------------------------------------------------*/
```

```
153     return;
154 }
155 
156 void fb_draw_rect(int x, int y, int w, int h, int color)
157 {
158     if(x < 0) { w += x; x = 0;}
159     if(x+w > SCREEN_WIDTH) { w = SCREEN_WIDTH-x;}
160     if(y < 0) { h += y; y = 0;}
161     if(y+h >SCREEN_HEIGHT) { h = SCREEN_HEIGHT-y;}
162     if(w<=0 || h<=0) return;
163     int *buf = _begin_draw(x,y,w,h);
164 /*-----------------------------------------------------*/
165     int x0, y0;
166     for(y0=y;y0<y+h;y0++){
167       for(x0=x;x0<x+w;x0++){
168         *(buf + y0*SCREEN_WIDTH + x0) = color;
169       }
170     }
171 /*-----------------------------------------------------*/
172     return;
173 }
174 
175 void fb_draw_line(int x1, int y1, int x2, int y2, int color)
176 {
177 /*-----------------------------------------------------*/
178     int xx, yy;
179     int stepx = (x2>x1)?1:-1;
180     int stepy = (y2>y1)?1:-1;
181     int* buf = _begin_draw(0,0,SCREEN_WIDTH,SCREEN_HEIGHT);
182     int slope = (x1==x2)?-1:abs((y2-y1)/(x2-x1));
183     if(slope<1 && slope>=0)
184     for(xx=x1;xx!=(x2+stepx);xx+=stepx){
185         yy = (y2-y1)*(xx-x1)/(x2-x1) + y1;
186         *(buf + yy*SCREEN_WIDTH + xx) = color;
187     }
188     else
189     for(yy=y1;yy!=(y2+stepy);yy+=stepy){
190         xx = (x2-x1)*(yy-y1)/(y2-y1) + x1;
191         *(buf + yy*SCREEN_WIDTH + xx) = color;
192     }
```

```
193
194 /*-------------------------------------------------*/
195     return;
196 }
197
198 void fb_draw_image(int x, int y, fb_image *image, int color)
199 {
200     if(image == NULL) return;
201
202     int ix = 0; //image x  实际绘图的图像起始坐标  而x.y代表屏幕上的显示范围
203     int iy = 0; //image y
204     int w = image->pixel_w; //draw width
205     int h = image->pixel_h; //draw height
206
207     if(x<0) {w+=x; ix-=x; x=0;}
208     if(y<0) {h+=y; iy-=y; y=0;}
209
210     if(x+w > SCREEN_WIDTH) {
211         w = SCREEN_WIDTH - x;
212     }
213     if(y+h > SCREEN_HEIGHT) {
214         h = SCREEN_HEIGHT - y;
215     }
216     if((w <= 0)||(h <= 0)) return;
217
218     int *buf = _begin_draw(x,y,w,h);
219 /*-----------------------------------------------------------------*/
220     //char *dst = (char *)(buf + y*SCREEN_WIDTH + x);
221     //char *src; //不同的图像颜色格式定位不同
222 /*-----------------------------------------------------------------*/
223
224     if(image->color_type == FB_COLOR_RGB_8880) /*lab3: jpg*/
225     {
226         int y0, y3;
227         char* dst = (char*)(buf+y*SCREEN_WIDTH+x);
228         char *src = image->content;
229         for(y0=y, y3=iy;y0<y+h;y0++,y3++){
230             /*for(x0=x, x3=ix;x0<x+w;x0++,x3++){
231                 temp = (int*)(image->content+y3*image->pixel_w*4+x3*4);
232                 *(buf+y0*SCREEN_WIDTH+x0) = *temp;
```

```
233             }*/
234             memcpy(dst,src,w*4);
235             dst += SCREEN_WIDTH*4;
236             src += w*4;
237         }
238 
239         return;
240     }
241     else if(image->color_type == FB_COLOR_RGBA_8888) /*lab3: png*/
242     {
243         //printf("you need implement fb_draw_image() FB_COLOR_RGBA_8888
    \n"); //exit(0);
244         int x0, y0, x3, y3;
245         unsigned char alpha;
246         char* colord, *temp;
247         for(y0=y, y3=iy;y0<y+h;y0++,y3++){
248             for(x0=x, x3=ix;x0<x+w;x0++,x3++){
249                 colord = (char*)(buf+y0*SCREEN_WIDTH+x0);
250                 temp = image->content+y3*image->pixel_w*4+x3*4;
251                 alpha = (unsigned char)(temp[3]);
252                 switch (alpha){
253                     case 0: break;
254                     case 255:
255                         colord[0] = temp[0];
256                         colord[1] = temp[1];
257                         colord[2] = temp[2];
258                         break;
259                     default:
260                         colord[0] += (((temp[0] - colord[0]) * alpha)
    >> 8);
261                         colord[1] += (((temp[1] - colord[1]) * alpha)
    >> 8);
262                         colord[2] += (((temp[2] - colord[2]) * alpha)
    >> 8);
263                 }
264             }
265         }
266         return;
267     }
268     else if(image->color_type == FB_COLOR_ALPHA_8) /*lab3: font*/
```

```
269     {
270         //printf("you need implement fb_draw_image() FB_COLOR_ALPHA_8
    \n");
271         int x0, y0, x3, y3;
272         unsigned char alpha;
273         char* colord, *temp;
274         for(y0=y, y3=iy;y0<y+h;y0++,y3++){
275             for(x0=x, x3=ix;x0<x+w;x0++,x3++){
276                 colord = (char*)(buf+y0*SCREEN_WIDTH+x0);
277                 temp = image->content+y3*image->pixel_w+x3;
278                 alpha = *temp;
279                 switch (alpha){
280                     case 0: break;
281                     case 255:
282                         colord[0] = (color & 0xff);
283                         colord[1] = (color & 0xff00)>>8;
284                         colord[2] = (color & 0xff0000)>>16;
285                         break;
286                     default:
287                         colord[0] += ((((color & 0xff) - colord[0]) *
    alpha) >> 8);
288                         colord[1] += (((((color & 0xff00)>>8)- colord
    [1]) * alpha) >> 8);
289                         colord[2] += (((((color & 0xff0000)>>16) - col
    ord[2]) * alpha) >> 8);
290                 }
291             }
292         }
293         return;
294     }
295 /*-------------------------------------------------------------------*/
296     return;
297 }
298
299 void fb_draw_border(int x, int y, int w, int h, int color)
300 {
301     if(w<=0 || h<=0) return;
302     fb_draw_rect(x, y, w, 1, color);
303     if(h > 1) {
304         fb_draw_rect(x, y+h-1, w, 1, color);
```

```
305         fb_draw_rect(x, y+1, 1, h-2, color);
306         if(w > 1) fb_draw_rect(x+w-1, y+1, 1, h-2, color);
307     }
308 }
309 
310 /** draw a text string **/
311 void fb_draw_text(int x, int y, char *text, int font_size, int color)
312 {
313     fb_image *img;
314     fb_font_info info;
315     int i=0;
316     int len = strlen(text);
317     while(i < len)
318     {
319         img = fb_read_font_image(text+i, font_size, &info);
320         if(img == NULL) break;
321         fb_draw_image(x+info.left, y-info.top, img, color);
322         fb_free_image(img);
323 
324         x += info.advance_x;
325         i += info.bytes;
326     }
327     return;
328 }
329 
330 /*-------------------------------------------------------------------------
    --------*/
331 
332 void fb_draw_circle(int x, int y, int r, int color)
333 {
334     int w, h, x0, y0, left, right, top, bottom;
335     if(x-r < 0) left = 0; else left = x-r;
336     if(x+r > SCREEN_WIDTH) right = SCREEN_WIDTH; else right = x+r;
337     if(y-r < 0) top = 0; else top = y-r;
338     if(y+r >SCREEN_HEIGHT) bottom = SCREEN_HEIGHT; else bottom = y+r;
339     w = right-left;
340     h = bottom-top;
341     x0 = left;
342     y0 = top;
343 
```

```
344     int *buf = _begin_draw(x0,y0,w,h);
345     for(y0=top;y0<=bottom;y0++){
346         for(x0=left;x0<=right;x0++){
347             double distance = sqrt((x0-x)*(x0-x)+(y0-y)*(y0-y));
348             if(distance <= r){
349                 *(buf+y0*SCREEN_WIDTH+x0) = color;
350             }
351         }
352     }
353     return;
354 }
355
356
357 void imagecentralize(imageplus* imgplus, fb_image* img){
358     imgplus->image = img;
359     imgplus->w = img->pixel_w;
360     imgplus->h = img->pixel_h;
361     imgplus->x = 512 - imgplus->image->pixel_w/2;
362     imgplus->y = 300 - imgplus->image->pixel_h/2;
363     fb_draw_image(imgplus->x, imgplus->y, imgplus->image, BLACK);
364     return;
365 }
366
367
368 void drawimage(imageplus* imgplus){
369     if(imgplus->image == NULL) return;
370
371     double ratio_incre = (double)(imgplus->image->pixel_w)/(imgplus->
    w);
372
373     int ix = 0; //image x  实际绘图的图像起始坐标  而x.y代表屏幕上的显示范围
374     int iy = 0; //image y
375     int w = imgplus->w; //draw width
376     int h = imgplus->h; //draw height
377     int x = imgplus->x;
378     int y = imgplus->y;
379
380     if(x<0) {w+=x; ix-=x; x=0;}
381     if(y<0) {h+=y; iy-=y; y=0;}
382
```

```
383     if(x+w > SCREEN_WIDTH) {
384         w = SCREEN_WIDTH - x;
385     }
386     if(y+h > SCREEN_HEIGHT) {
387         h = SCREEN_HEIGHT - y;
388     }
389     if((w <= 0)||(h <= 0)) return;
390 
391     int *buf = _begin_draw(x,y,w,h);
392     int *temp;
393 
394     //if(imgplus->image->color_type == FB_COLOR_RGB_8880)
395     //{
396         double x3, y3;
397         int x0, y0;
398         for(y0=y, y3=iy;y0<y+h;y0++){
399             for(x0=x, x3=ix;x0<x+w;x0++){
400                 temp = (int*)(imgplus->image->content+((int)y3)*imgplus
    ->image->pixel_w*4+((int)x3)*4);
401                 *(buf+y0*SCREEN_WIDTH+x0) = *temp;
402                 x3 += ratio_incre;
403             }
404             y3 += ratio_incre;
405         }
406     //}
407     return;
408 }
409 
410 
411 void imagescaling(imageplus* imgplus, int type){
412     int zoom_size_x = 50;
413     int zoom_size_y;
414     int offset_size = 60;
415     switch(type){
416         case 0://fangda
417             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
418             zoom_size_y = (int)(zoom_size_x * ((double)(imgplus->image-
    >pixel_h)/(imgplus->image->pixel_w)));
419             imgplus->x -= zoom_size_x;
420             imgplus->y -= zoom_size_y;
```

```
421             imgplus->w += 2*zoom_size_x;
422             imgplus->h += 2*zoom_size_y;
423             drawimage(imgplus);
424             fb_draw_rect(0,0,100,100,PURPLE);
425             break;
426         case 1://suoxiao
427             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
428             zoom_size_y = (int)(zoom_size_x * ((double)(imgplus->image-
    >pixel_h)/(imgplus->image->pixel_w)));
429             imgplus->x += zoom_size_x;
430             imgplus->y += zoom_size_y;
431             imgplus->w -= 2*zoom_size_x;
432             imgplus->h -= 2*zoom_size_y;
433             drawimage(imgplus);
434             fb_draw_rect(0,0,100,100,PURPLE);
435             break;
436         case 2://zuoyi
437             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
438             imgplus->x -= offset_size;
439             drawimage(imgplus);
440             fb_draw_rect(0,0,100,100,PURPLE);
441             break;
442         case 3://youyi
443             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
444             imgplus->x += offset_size;
445             drawimage(imgplus);
446             fb_draw_rect(0,0,100,100,PURPLE);
447             break;
448         case 4://shangyi
449             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
450             imgplus->y -= offset_size;
451             drawimage(imgplus);
452             fb_draw_rect(0,0,100,100,PURPLE);
453             break;
454         case 5://xiayi
455             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
456             imgplus->y += offset_size;
457             drawimage(imgplus);
458             fb_draw_rect(0,0,100,100,PURPLE);
459             break;
```

```
460         case 6://quanping
461             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
462             imgplus->w = SCREEN_WIDTH;
463             imgplus->h = (int)(SCREEN_WIDTH * ((double)(imgplus->image-
    >pixel_h)/(imgplus->image->pixel_w)));
464             imgplus->x = 0;
465             imgplus->y = 0;
466             drawimage(imgplus);
467             fb_draw_rect(0,0,100,100,PURPLE);
468             break;
469         case 7://huifu
470             fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT,BLACK);
471             imagecentralize(imgplus, imgplus->image);
472             fb_draw_rect(0,0,100,100,PURPLE);
473             break;
474         default:
475             break;
476     }
477     return;
478 }
```

## Deepseek 对于 graphic.c 原理的阐释

在 Linux 的虚拟化环境中，上述代码通过直接操作帧缓冲区（framebuffer）来在屏幕上绘制各种图形和图像。以下是该代码的工作原理及其在虚拟化环境中的应用的详细解释：

### 帧缓冲区操作

1. **初始化帧缓冲区**：
   - 通过 `fb_init` 函数打开帧缓冲设备，并将其设置为图形模式。
   - 使用 `mmap()` 将帧缓冲区映射到内存，使程序可以直接访问和修改屏幕的像素数据。
2. **绘图函数**：
   - 函数如 `fb_draw_pixel`、`fb_draw_rect`、`fb_draw_line` 和 `fb_draw_image` 在 `DRAW_BUF` 缓冲区中操作像素数据，该缓冲区保存要应用于屏幕的更改。
   - `update_area` 结构体跟踪屏幕的修改部分，以便仅更新帧缓冲区中的这些区域，从而提高效率。
3. **更新帧缓冲区**：

- `fb_update` 函数将 `DRAW_BUF` 的内容复制到实际的帧缓冲区内存，仅更新 `update_area` 指定的区域，减少数据复制量，提高性能。

## 虚拟化环境中的操作

1. **虚拟设备交互**：
   - 在虚拟化环境中， guest OS 通过 hypervisor 提供的虚拟设备与硬件交互。
   - 当代码在虚拟机中运行时，帧缓冲区操作由 hypervisor 处理，将其转换为物理硬件操作。
2. **内存映射**：
   - `mmap()` 调用在虚拟环境中将 guest 的虚拟内存映射到 hypervisor 管理的帧缓冲区内存。
   - 当代码写入 `DRAW_BUF` 时，实际上是在写入 hypervisor 控制的内存区域，hypervisor 负责更新物理屏幕。
3. **绘图函数**：
   - 绘图函数在 `DRAW_BUF` 中操作，hypervisor 负责将这些更改同步到物理显示。
   - 这种抽象允许 guest OS 像直接访问硬件一样操作，而 hypervisor 管理实际的硬件交互。

## 性能考虑

1. **最小化更新**：
   - 通过跟踪 `update_area` ，代码确保仅更新必要的屏幕部分，减少虚拟化环境中的开销。
2. **图像格式处理**：
   - `fb_draw_image` 函数处理不同图像格式（如 RGB8880、RGBA8888、Alpha8），并考虑透明度进行混合。
   - 图像缩放和位置调整由 `imagescaling` 和 `drawimage` 函数处理。
3. **虚拟化开销**：
   - 在虚拟化环境中，缩放操作可能由于 hypervisor 的介入而更加耗资源。
   - 高效的内存管理和绘图算法对于保持性能至关重要。

## 总结

该代码通过直接操作帧缓冲区内存提供了低级别的图形绘制功能。在虚拟化环境中，这些操作由 hypervisor 管理，hypervisor 模拟帧缓冲设备并处理实际的渲染工作。通过最小化更新和高效的绘图算法，代码在虚拟化环境中实现了流畅的图形渲染。

# 实验四

## 实验四准备

连接触摸板，并且同时连接 hdmi、power 和 touch 到开发板

实验四的主要任务是补充 main.c 中对于触摸信号的响应代码 touch_event_fb 中的内容。

## 实验四开发过程

直接连接触摸板运行了 lab4 试试，看到标准输出有报错

```
root@orangepi4-lts:/lab-2022-st/out# ./lab4
framebuffer info: bits_per_pixel=32, size=(1024,600), virtual_pos_size=(0,0)(1024,600), line_length=4096, smem_len=245
7600
touch_init open /dev/input/event2 error!errno = 2
error: fd=-1, callback=0xaaaab17135f0
^Z
[2]+  Stopped                 ./lab4
```

```c
int main(int argc, char *argv[])
{
    fb_init("/dev/fb0");
    fb_draw_rect(x: 0,y: 0,w: SCREEN_WIDTH,h: SCREEN_HEIGHT,
    color: COLOR_BACKGROUND);
    fb_update();

    //打开多点触摸设备文件，返回文件fd
    touch_fd = touch_init("/dev/input/event1");
    //添加任务，当touch_fd文件可读时，会自动调用touch_event_cb函数
    task_add_file(touch_fd, callback: touch_event_cb);

    task_loop(); //进入任务循环
    return 0;
}
```

改成 event1 以后不报错了，但是摸上去没反应，应该是因为实现代码没写

运行以后发现摸上去还是没反应，再改成 event2 试试

```
root@orangepi4-lts:/lab-2022-st/out# ./lab4
framebuffer info: bits_per_pixel=32, size=(1024,600), virtual_pos_size=(0,0)(1024,600), line_length=4096, smem_len=245
7600
TOUCH_PRESS: x=471,y=350,finger=0
TOUCH_RELEASE: x=471,y=350,finger=0
TOUCH_PRESS: x=418,y=331,finger=0
TOUCH_RELEASE: x=418,y=331,finger=0
TOUCH_PRESS: x=407,y=327,finger=0
TOUCH_RELEASE: x=407,y=327,finger=0
TOUCH_PRESS: x=532,y=379,finger=0
TOUCH_PRESS: x=253,y=316,finger=1
TOUCH_PRESS: x=133,y=392,finger=2
TOUCH_RELEASE: x=532,y=379,finger=0
TOUCH_RELEASE: x=253,y=316,finger=1
TOUCH_RELEASE: x=133,y=392,finger=2
```

现在有反应了，破案了。之前报错是因为没有连触摸屏的 touch 接口。