***2020***



**嵌入式系统 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1808 |
| 学 号： | U201814719 |
| 姓 名： | 马忠平 |
| 同组人： | 陈文扬 |
| 电 话： | 18315025187 |
| 邮 件： | [1814729186@qq.com](mailto:2432152552@qq.com) |
| 完成日期： | 2021-6-18 |



目 录

[1 Linux内核编译和系统烧录 1](#_Toc75267339)

[1.1 实验要求 1](#_Toc75267340)

[1.2 方案设计 1](#_Toc75267341)

[1.3 结果验证 3](#_Toc75267342)

[1.4 故障与调试 3](#_Toc75267343)

[2 Linux framebuffer界面显示开发 5](#_Toc75267344)

[2.1 实验要求 5](#_Toc75267345)

[2.2 方案设计 5](#_Toc75267346)

[2.3 结果验证 9](#_Toc75267347)

[2.4 故障与调试 9](#_Toc75267348)

[3 图片显示和文本显示 11](#_Toc75267349)

[3.1 实验要求 11](#_Toc75267350)

[3.2 方案设计 11](#_Toc75267351)

[3.3 结果验证 15](#_Toc75267352)

[3.4 故障与调试 16](#_Toc75267353)

[4 Linux 多点触摸开发 17](#_Toc75267354)

[4.1 实验要求 17](#_Toc75267355)

[4.2 方案设计 17](#_Toc75267356)

[4.3 结果验证 21](#_Toc75267357)

[4.4 故障与调试 22](#_Toc75267358)

[5 蓝牙无线互联通讯 23](#_Toc75267359)

[5.1 实验要求 23](#_Toc75267360)

[5.2 方案设计 23](#_Toc75267361)

[5.3 结果验证 24](#_Toc75267362)

[6 综合设计——联机五子棋游戏 26](#_Toc75267363)

[6.1 实验要求 26](#_Toc75267364)

[6.2 方案设计 26](#_Toc75267365)

[6.3 结果验证 34](#_Toc75267366)

[6.4 故障与调试 37](#_Toc75267367)

[7 总结与心得 39](#_Toc75267368)

[7.1 实验总结 39](#_Toc75267369)

[7.2 实验心得 39](#_Toc75267370)

[8 附录 40](#_Toc75267371)

[8.1 图形库代码 40](#_Toc75267372)

[8.2 联机五子棋游戏代码 45](#_Toc75267373)

[参考文献 57](#_Toc75267374)

# Linux内核编译和系统烧录

## 实验要求

本实要求使用Linux内核源码进行内核编译，生成系统镜像并将系统烧录到开发板上。学习使用串口工具cutecom控制uboot，使用USB烧写工具fastboot烧录核心镜像和文件系统。使用Android NDK交叉编译简单应用程序，并使用usb开发工具adb上传应用到开发板，运行程序进行测试。

## 方案设计

### 内核编译

本项实验将使用linux-3.0.15版本内核，目标为arm架构，下载内核后进入内核主目录，执行“make menuconfig”命令配置内核生成“.config”文件；执行“make zImage”命令生成gzip压缩镜像zImage，该镜像为Linux内核，其中包含操作系统的调度、通信等核心模块以及配置为编译到内核的各种驱动，zImage镜像输出路径为“内核主目录/arch/arm/boot/zImage”。

为方便内核烧录验证，我们应确保“内核主目录/include/generated”目录下不存在“compile.h”文件；该文件将在编译内核时自动生成，其中记录了内核生成时间，烧录完成后可通过“uname -a”查看。

### 内核烧录

配置USB设备，在/etc/udev/rules.d/下创建配置文件51-android.rules，并填入如下内容。

|  |
| --- |
| SUBSYSTEM==”usb”,ATTR(idVendor)==”18d1”,MODE=”0666”,GROUP=”plugdev”,OWNER=”hust” |

SUBSYSTEM标识这是一个USB设备；ATTR(idVendor)是开发板厂商的ID号；MODE为访问权限，0666表示该设备对所有用户组均可读写；GROUP配置该设备为即插即用模式；OWNER指定拥有操作权限的用户。

在用户根目录下创建adb\_usb.ini文件，并写入主设备号“0x18d1”。以管理员权限运行cutecom，配置设备为“dev/tty50”，波特率为115200，数据位宽为8，串口结束位为1，允许读写。

连接串口线、USB线到开发板，接通开发板电源，开机。此时在cutecom界面打开设备，发送任意信息以中断引导。cutecom界面输入“fastboot”命令，继续输入如下命令依次烧录uboot、Linux内核、根文件系统及主文件系统（假定当前已处于存放镜像的目录下）。

|  |
| --- |
| fastboot flash bootloader u-boot.bin  fastboot flash kernel zImage  fastboot flash ramdisk ramdiskuboot.img  fastboot -w  fastboot flash system system.img |

### 简单应用程序开发

使用C语言编写一个简单的Hello World程序。

|  |
| --- |
| #include <stdio.h>  int main(int argc, char \*argv[]) {  printf(“Hello embedded linux!(CWY&MZP)\n”);  return 0;  } |

使用安卓android-ndk-r8c开发工具包交叉编译连接上述程序，生成可执行程序，命名为lab1。查看lab1的ELF头，可得如图 1.1所示信息，它表明该程序以ARM架构的机器为目标平台，运行在基于UNIX的操作系统上。

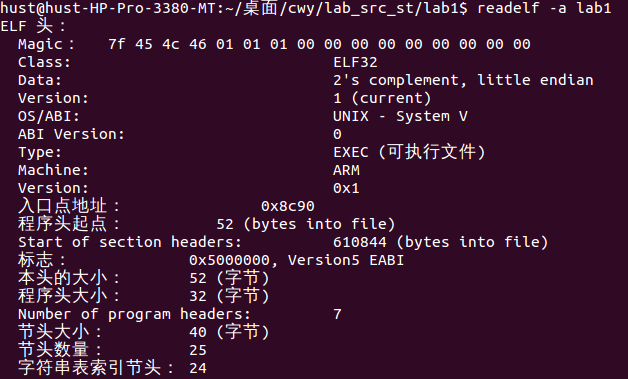


图 1.1 程序交叉编译结果

## 结果验证

开发板通电后开机，待系统引导完成后，执行命令“uname -a”查看内核信息，可得如图 1.2所示结果，编译时间表明这是一个新编译的内核。



图 1.2 内核信息

将lab1程序上传到开发板目录。

|  |
| --- |
| adb push lab1 /data/local/  adb shell chmod +x /data/local/lab1 |

运行lab1，可得如图 1.3所示输出结果，该程序能够正常运行。



图 1.3 lab1执行结果

## 故障与调试

### 操作系统引导失败导致循环重启

故障现象：烧录内核镜像到开发板后按下复位按钮，从串口工具cutecom中发现系统未能正常运行，并且在失败后循环重启。

原因分析：内核的模块配置不完整，内核生成规则“.config”文件配置的编译到内核模块缺少一系列项目，导致系统运行缺少必要的组件。

解决方案：使用“make menuconfig”重新配置内核生成规则“.config”，将必要组件全部编译到内核。

# Linux framebuffer界面显示开发

## 实验要求

了解Linux下的LCD显示驱动接口framebuffer的使用原理。

编写绘图函数，实现点、线、矩形区域等基本图形的显示。

了解双缓冲机制及其工作流程。

## 方案设计

### 基本数据结构和工具函数

本项实验提供一部分已经实现的数据结构、工具函数和宏。

后备显示缓冲是一个规模为1024\*600的int型全局数组，update\_area用于修改更新区域，这一区域的像素数据将从后备显示缓冲拷贝到framebuffer。

工具函数及其描述如表 2.1所示。

表 2.1 工具函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| fb\_init | 初始化程序与LCD显示器的连接，包括打开设备文件和建立图形缓冲到程序上下文的映射。 |
| \_copy\_area | 从后备缓冲拷贝绘制区像素信息到LCD显示缓冲。 |
| \_check\_area | 裁剪绘制区，去除不可见部分。 |
| fb\_update | 拷贝绘制区像素信息以更新LCD显示缓冲。 |
| \_begin\_draw | 更新绘制区域并返回后备缓冲起始指针。 |
| fb\_draw\_pixel | 绘制点。 |

### 矩形绘制函数

绘制矩形的方法是从矩形左上角端点开始从左到右从上到下扫描填充像素，直至到达右下角端点。其实现如下。

|  |
| --- |
| void fb\_draw\_rect(int x, int y, int w, int h, int color)  {  if(x < 0) { w += x; x = 0;}  if(x+w > SCREEN\_WIDTH) { w = SCREEN\_WIDTH-x;}  if(y < 0) { h += y; y = 0;}  if(y+h >SCREEN\_HEIGHT) { h = SCREEN\_HEIGHT-y;}  if(w<=0 || h<=0) return;  int \*buf = \_begin\_draw(x,y,w,h);  int i, j;  for (i = 0; i < h; ++i) {  for (j = 0; j < w; ++j) {  buf[(i + y) \* SCREEN\_WIDTH + j + x] = color;  }  }  } |

### 单像素线条绘制函数

单像素线条绘制要求线条必须连续不断开，即线条像素邻点间的横纵坐标至多相差1。此处采用Bresenham直线算法实现线条绘制，主要步骤如下：

①计算线的两个端点的横、纵坐标差dx、dy，取max{dx, dy}的两倍为步进槽高dh。

②预填充步进槽为一半高度。

③以端点坐标差较小的坐标轴为步进轴，从步进轴坐标较小的端点开始沿步进轴方向绘制像素点，并填充步进槽1个单位，当步进槽填满时，沿非步进轴向终端点移进1个单位，清空步进槽后。

④重复③过程，直至到达终端点。

单像素线条绘制函数实现如下。

|  |
| --- |
| void fb\_draw\_line(int x1, int y1, int x2, int y2, int color) {  int dx, // difference in x's  dy, // difference in y's  dx2, // dx,dy \* 2  dy2,  x\_inc, // amount in pixel space to move during drawing  y\_inc, // amount in pixel space to move during drawing  error, // the discriminant i.e. error i.e. decision variable  index; // used for looping  // compute horizontal and vertical deltas  dx = x2 - x1;  dy = y2 - y1;  // test which direction the line is going in i.e. slope angle  if (dx >= 0) {  x\_inc = 1;  } // end if line is moving right  else {  x\_inc = -1;  dx = -dx; // need absolute value  } // end else moving left  // test y component of slope  if (dy >= 0) {  y\_inc = 1;  } // end if line is moving down  else {  y\_inc = -1;  dy = -dy; // need absolute value  } // end else moving up  // compute (dx,dy) \* 2  dx2 = dx << 1;  dy2 = dy << 1;  int beg\_x = x1 < x2 ? x1 : x2;  int beg\_y = y1 < y2 ? y1 : y2;  int\* buf = \_begin\_draw(beg\_x, beg\_y, dx, dy);  int x = x1, y = y1; //indexes for ploting pixels  // now based on which delta is greater we can draw the line  if (dx > dy) {  error = dy2 - dx; // initialize error term  // draw the line  for (index = 0; index <= dx; index++) {  fb\_draw\_pixel(x ,y, color); // set the pixel  if (error >= 0) { // test if error has overflowed  error -= dx2;  y += y\_inc; // move to next line  } // end if error overflowed  error += dy2; // adjust the error term  x += x\_inc; // move to the next pixel  } // end for  } // end if |k| <= 1  else {  error = dx2 - dy; // initialize error term  // draw the line  for (index = 0; index <= dy; index++) {  fb\_draw\_pixel(x ,y, color); // set the pixel  if (error >= 0) { // test if error overflowed  error -= dy2;  x += x\_inc; // move to next line  } // end if error overflowed  error += dx2; // adjust the error term  y += y\_inc; // move to the next pixel  } // end for  } // end else |k| > 1  } |

## 结果验证

将点、线、矩形绘制函数集成到测试代码进行交叉编译，生成可执行文件lab2。将测试程序lab2上传到开发板运行，显示效果如图 2.1所示。

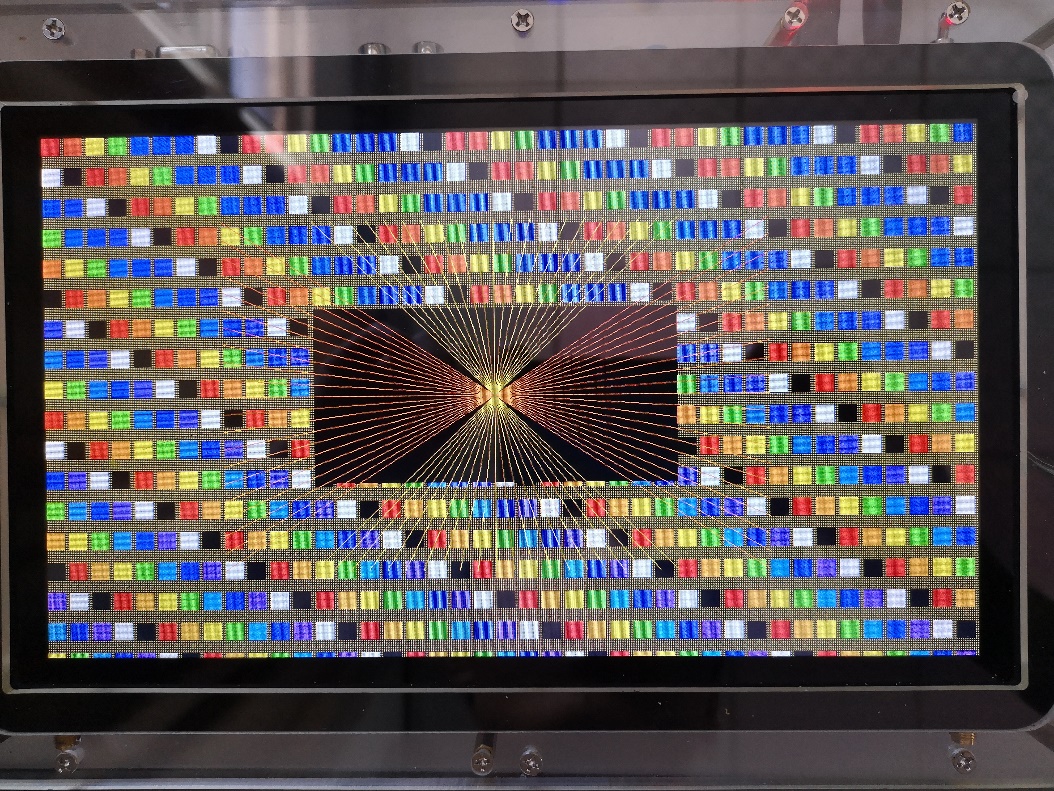


图 2.1 点、线、矩形绘制测试程序显示效果

## 故障与调试

### 后备缓冲写入地址越界导致线条绘制出错

故障现象：程序执行异常退出，系统报段错误，LCD显示屏无线条显示。



图 2.2 线条绘制出错

原因分析：线条绘制函数中的下述代码试图获取线条起始端点像素在后备缓冲中的地址，但“\_begin\_draw”的返回值为void\*类型，编译器按照默认规则按照char类型大小执行指针加法，而实际上应当按照int类型执行指针加法，最终导致buf指针值不符预期。

|  |
| --- |
| int\* buf = \_begin\_draw(beg\_x, beg\_y, dx, dy) + y1 \* SCREEN\_WIDTH + x1; |

解决方案：使用强制类型转换，或者推迟到填充像素点时计算偏移。

|  |
| --- |
| int\* buf = (int\*)\_begin\_draw(beg\_x, beg\_y, dx, dy) + y1 \* SCREEN\_WIDTH + x1; |

# 图片显示和文本显示

## 实验要求

本实要求实现图片显示和矢量字体显示。

要求应用能够识别jpg和png格式的图片，能够处理透明度。

要求应用能够提取矢量字模并完成字模的显示。

## 方案设计

### 基本数据结构和相关工具函数

本项实验为使用提供的工具函数，封装相关代码实现图片和文本的显示，已经提供的工具函数如表3. 1所示。

表3. 1 工具函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| fb\_read\_jpeg\_image | 从文件获取jpeg格式的图片，将图片信息封装成结构体并返回结构体指针。 |
| fb\_read\_png\_image | 从文件获取png格式的图片，将图片信息封装成结构体并返回结构体指针。 |
| fb\_read\_font\_image | 获取字模信息，返回结构体指针。 |
| fb\_get\_sub\_image | 获取一个图片的子图片。 |
| fb\_free\_image | 释放图片文件信息。 |

执行fb\_read\_jpeg\_image、fb\_read\_png\_image、fb\_get\_sub\_image函数得到的数据结构定义如下。

|  |
| --- |
| typedef struct {  int color\_type; /\*图片中的像素颜色类型\*/  int pixel\_w, pixel\_h; /\*图片的像素宽和高\*/  int line\_byte; /\*图片中存储一行像素颜色所占的字节数\*/  char \*content; /\*图片的像素颜色值\*/  } fb\_image; |

对于不透明格式jpg图片而言，图片不存在透明通道，每个像素点用32字节储存像素信息，从低到高0-7位储存蓝色信息，8-15位储存绿色通道值，16-23位储存红色通道值。对于png格式图片而言，低24位储存信息与jpg一致，高8位储存透明通道信息。对于字模数据，每个像素位点用一字节储存透明度信息Alpha值。在代码编写过程中，需要针对三种情况作不同处理。

### jpg不透明图片绘制

对于不透明图片，仅需要将数据信息按行拷贝到显示缓冲中即可。需要注意的是，这里需要对图片显示超出屏幕边界的情况进行处理：为避免内存泄漏，需要将图片宽度和当前起始坐标下屏幕可用宽度进行判断，用较小的值作为拷贝数据到缓冲区的行宽，并在拷贝完成之后，分别按图片行和屏幕行进行偏移。

jpg不透明图片的显示处理代码块如下。

|  |
| --- |
| if(image->color\_type == FB\_COLOR\_RGB\_8880) /\*lab3: jpg\*/  {  /\*get a line of colors and copy them to buffer\*/  int length = image->line\_byte;  if(image->line\_byte > 4 \* w) length = w\*4; /\*close to the border\*/  for(int i=0;i<h;i++){  memcpy(dst,src,length);  dst += SCREEN\_WIDTH; //shift  src += image->line\_byte; //shift  }  return;  } |

### png半透明图片显示

对于半透明图片信息，在jpg图片需要的注意点之上，还需要对图片信息的alpha通道进行处理。最终颜色color = color1 \* alpha1 + color2 \* (1 - alpha1)。

png半透明图片的显示处理代码块如下。

|  |
| --- |
| if(image->color\_type == FB\_COLOR\_RGBA\_8888) /\*lab3: png\*/  {  for(int i = 0;i < w,i++){  for(int j = 0;j < h;j++){  //当前点的显存地址信息  int \* addr = buf + (i+x)\*SCREEN\_WIDTH + y + j;  char \*p = (char\*)addr;  //获得当前点的RGB样式  int argb = \*addr;  //图像信息  int pic\_argb = \*(image->content + i\*image->line\_byte/4 + j);  alpha = pic\_rgba >> 24;  R1 = (argb & 0xff0000)>>16;  G1 = (argb & 0xff00)>>8;  R1 = argb & 0xff;  switch(alpha){  case 0:break;  case 255:\*(addr) = pic\_argb;break;  default:  p[0] += (((B1 - p[0]) \* alpha) >> 8);  p[1] += (((G1 - p[1]) \* alpha) >> 8);  p[2] += (((R1 - p[2]) \* alpha) >> 8);  }  }  }  return;  } |

### 矢量字体显示

对于矢量字体显示，仅需提取alpha字段并对其进行处理即可。rgb颜色从实参中获取。矢量字体显示的处理代码如下。

|  |
| --- |
| if(image->color\_type == FB\_COLOR\_ALPHA\_8) /\*lab3: font\*/  {  R1 = (color&0xff0000)>>16;  G1 = (color&0xff00)>>8;  B1 = color&0xff;  //字模  for(int i = 0;i < h;i++){  for(int j = 0;j < w;j++){  int\* addr = buf + (i + x) \* SCREEN\_WIDTH + y + j;  char\* p = (char\*)addr;  //获得Alpha  alpha = \*(char\*)(image->content + i \* image->line\_byte + j);  switch (alpha) {  case 0:break;  case 255:\*(addr) = pic\_argb; break;  default:  p[0] += (((B1 - p[0]) \* alpha) >> 8);  p[1] += (((G1 - p[1]) \* alpha) >> 8);  p[2] += (((R1 - p[2]) \* alpha) >> 8);  }  }  }  return;  } |

## 结果验证

将图片、字体绘制函数测试程序进行交叉编译，生成可执行文件lab3，将可执行文件和相关资源文件上传到平台并运行，显示效果如图3. 1所示。

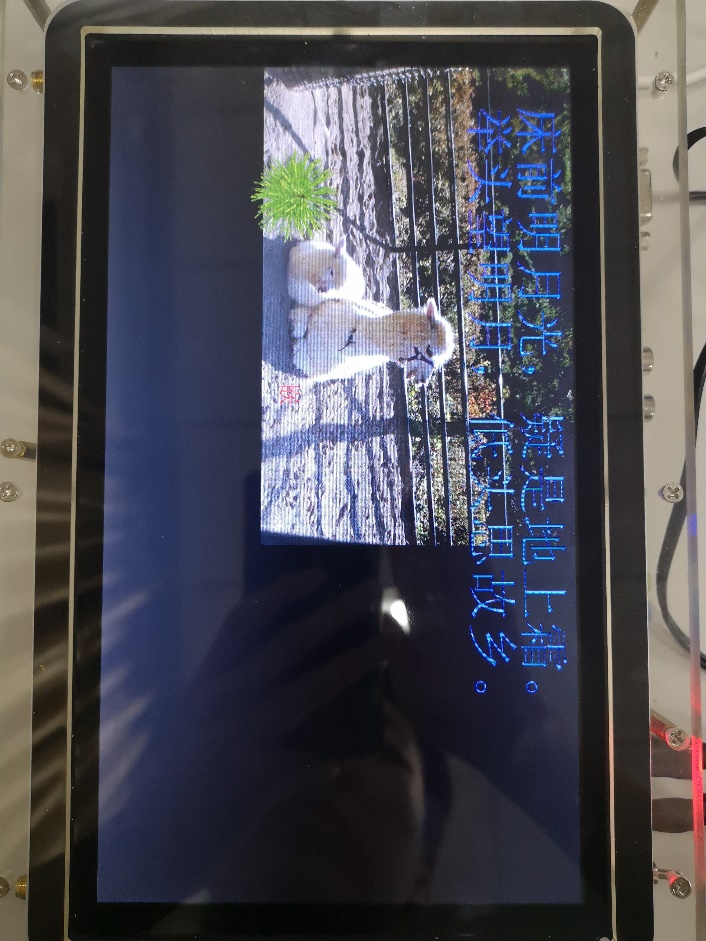


图3. 1 测试文件运行结果

可见不透明图片、透明图片、字模信息均能正常显示。

将test项目测试程序进行交叉编译，生成可执行文件test，将可执行文件和相关资源文件长传到平台并运行常测试，得到的运行结果如图3. 2所示。

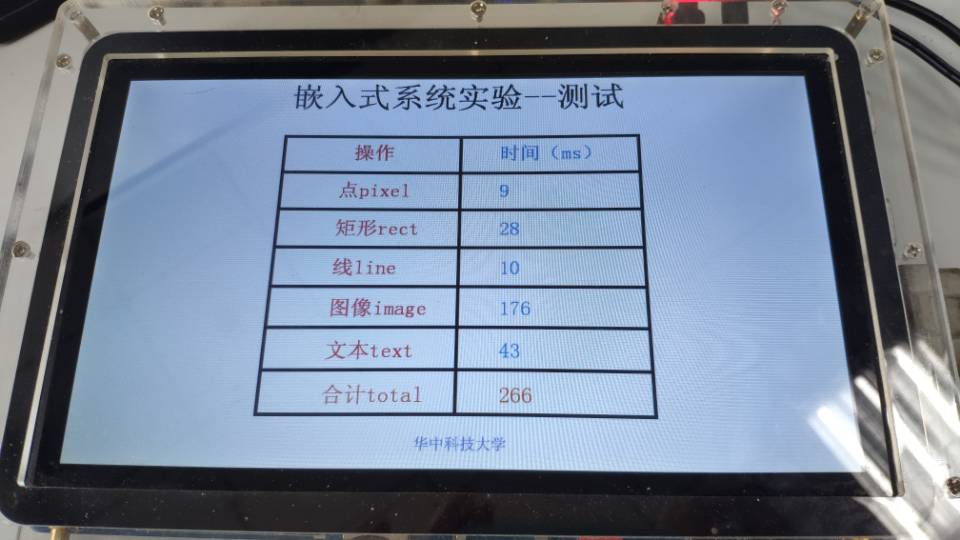


图3. 2 测试程序运行结果

## 故障与调试

### 图片绘制出错

**故障现象：**在执行lab3测试程序时，显示出的图片混乱，无法显示出正确的图片信息。

**原因分析：**循环体中像素点处理代码x、y轴处理逻辑弄反了。且因为使用的是char\*指针，每个单位的指针偏移仅偏移1字节，而设计目的为偏移4字节，即一个像素点。

**解决方案：**重构代码，将x、y轴分别处理，外层循环为x轴坐标偏移量，内层循环为y轴坐标偏移量，使用int\*指针处理相关信息，使得每个单位偏移量对应于图片信息中的一个像素点。

# Linux 多点触摸开发

## 实验要求

本实要求学习Linux下的触摸屏驱动接口，包括Input event的使用及多点触摸协议（Multi-touch Protocol）。

编写能够实现以下功能的应用程序：

①获取多点触摸的坐标。

②在LCD上显示多点触摸轨迹。

③绘制一个清除屏幕的按钮，点击后清除屏幕内容。

## 方案设计

### 绘制粗线条

考虑到显示效果，触摸屏中绘制的线条应当具有清晰的轨迹。我们用以下两个函数来绘制粗线条。

|  |
| --- |
| //扩展fb\_draw\_line画线函数的功能，沿着线条轨迹调用画笔函数pen  void fb\_draw\_line\_pen(int x1, int y1, int x2, int y2, void (\*pen)(int x, int y, int color), int color);  void dot\_rx(int x, int y, int color); //绘制以(x,y)为圆心的圆，其半径rx由全局变量指定 |

我们将dot\_rx作为fb\_draw\_line\_pen的画笔，便可画出宽度为2rx的粗线条。

其中，fb\_draw\_line\_pen由fb\_draw\_line修改而来，仅将函数体中对fb\_draw\_pixel的调用替换成对画笔函数pen的调用。

dot\_rx采用中点画圆算法实现，仅需计算八分之一圆周即可，其他圆周点通过对称性得出；将左右对称点连接，绘制线条以填充圆域。画圆函数实现如下。

|  |
| --- |
| static int rx = 8; // 画圆半径  static void dot\_rx(int x, int y, int color) {  int tx = 0, ty = rx, d = 3 - 2 \* rx;  while (tx < ty) { // 小于 45 度横线  fb\_draw\_line(x - ty, y - tx, x + ty, y - tx, color);  if (tx != 0) { // 防止水平线重复绘制  fb\_draw\_line(x - ty, y + tx, x + ty, y + tx, color);  }  if (d < 0) { // 取上面的点  d += 4 \* tx + 6;  }  else { // 取下面的点  // 大于 45 度横线  fb\_draw\_line(x - tx, y - ty, x + tx, y - ty, color);  fb\_draw\_line(x - tx, y + ty, x + tx, y + ty, color);  d += 4 \* (tx - ty) + 10; ty--;  }  tx++;  }  if (tx == ty) { // 45 度横线  fb\_draw\_line(x - ty, y - tx, x + ty, y - tx, color);  fb\_draw\_line(x - ty, y + tx, x + ty, y + tx, color);  }  } |

### 按钮响应

按钮事件分两步进行处理，首先判定触发按钮，然后调用触发按钮的处理过程。简单起见，我们采用最朴素的矩形区域判定方法来识别触发按钮。本项实验中，我们设计了2个按钮，其一用于变更画笔粗细程度，另一个用于清空屏幕。

|  |
| --- |
| enum { BTN\_CLS = 1, BTN\_CHG }; //1-清空屏幕，2-变更画笔  static int BtnHit(int x, int y) { //矩形区域按钮触发判定  if (x >= 50 && x <= 100 && y >= 50 && y <= 100) { return BTN\_CHG; }  if (x >= 50 && x <= 100 && y >= 150 && y <= 200) { return BTN\_CLS; }  return 0; //零值表示没有按钮被触发  } |

两个按钮事件的处理流程如表 4.1所示。

表 4.1 按钮事件处理流程

|  |  |
| --- | --- |
| 按钮事件 | 处理流程 |
| BTN\_CLS 清空屏幕 | 重绘画布、按钮及画笔粗号数字 |
| BTN\_CHG 变更画笔 | 增加画笔粗度，如果超过最大值则重置为1，重绘画笔粗号数字 |

根据前述事件处理流程将按钮事件处理流程实现如下。

|  |
| --- |
| static void BtnCall(int cmd) { //按钮事件响应  char buf[32]; fb\_image \*img;  switch(cmd) {  case BTN\_CLS: //清空屏幕  //重绘画布  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,COLOR\_BACKGROUND);  fb\_draw\_rect(40,40,70,5,COLOR\_RED);  fb\_draw\_rect(35,40,5,50,COLOR\_RED);  img = fb\_read\_png\_image("/data/local/clear.png"); //重绘按钮  fb\_draw\_image(50, 150, img, 0);  if (rx >= 10) { buf[0] = rx / 10 + '0'; buf[1] = rx % 10 + '0'; buf[2] = '\0'; }  else { buf[0] = rx + '0'; buf[1] = '\0'; }  fb\_draw\_text(50, 99, buf, 64, COLOR\_BROWN);  fb\_update();  fb\_free\_image(img);  break;  case BTN\_CHG: //变更画笔  rx >= 66 ? (rx = 1) : (++rx); //如果粗号超过66则重置为1  if (rx >= 10) { buf[0] = rx / 10 + '0'; buf[1] = rx % 10 + '0'; buf[2] = '\0'; }  else { buf[0] = rx + '0'; buf[1] = '\0'; }  //重绘按钮和粗号数字  fb\_draw\_rect(40,40,75,60,COLOR\_BACKGROUND);  fb\_draw\_rect(40,40,70,5,COLOR\_RED);  fb\_draw\_rect(35,40,5,50,COLOR\_RED);  fb\_draw\_text(50, 99, buf, 64, COLOR\_BROWN);  fb\_update();  break;  default: break;  }  } |

### 触摸屏事件响应

我们主要关注对三个触摸屏事件的响应，处理流程如表 4.2所示。

表 4.2 触摸屏事件处理流程

|  |  |
| --- | --- |
| 触摸屏事件 | 处理流程 |
| TOUCH\_PRESS 按下 | 记录触发事件分配的slot编号，x、y坐标值，分配一个随机颜色 |
| TOUCH\_MOVE 移动 | 根据slot编号，从上一个坐标向当前坐标绘制线条，并记录当前坐标 |
| TOUCH\_RELEASE 释放 | 如果slot编号与记录slot相同，则进行按钮响应处理 |

为支持触摸屏事件响应，提供如下存储量。

|  |
| --- |
| static int oldX[5]; //记录5个slot的横坐标旧值  static int oldY[5]; //记录5个slot的纵坐标旧值  static int color[5]; //记录为5个slot分配的色彩值  static int oldFinger; //记录上一个slot编号  int type, ; //暂存事件类型  x,y, //暂存当前slot的横纵坐标值  finger; //暂存当前slot编号 |

根据前述事件处理流程将触摸屏事件处理流程实现如下。

|  |
| --- |
| static void touch\_event\_cb(int fd)  {  static int oldX[5] = { 0, 0, 0, 0, 0 };  static int oldY[5] = { 0, 0, 0, 0, 0 };  static int color[5];  static int oldFinger = -1;  int type,x,y,finger;  type = touch\_read(fd, &x,&y,&finger);  switch(type) {  case TOUCH\_PRESS:  oldFinger = finger; oldX[finger] = x; oldY[finger] = y;  color[finger] = FB\_COLOR((rand() & 0xff), (rand() & 0xff), (rand() & 0xff));  break;  case TOUCH\_MOVE:  fb\_draw\_line(oldX[finger], oldY[finger], x, y, color[finger]);  oldX[finger] = x; oldY[finger] = y;  break;  case TOUCH\_RELEASE:  if (oldFinger == finger) { BtnCall(BtnHit(x, y)); }  break;  case TOUCH\_ERROR:  printf("close touch fd\n"); close(fd); task\_delete\_file(fd);  break;  default: return;  }  fb\_update();  } |

## 结果验证

将新增的绘制函数添加到图形库，集成事件响应处理模块进行交叉编译，生成可执行文件lab4。将多点触摸演示程序lab4上传到开发板运行，显示效果如图 4.1、图 4.2所示。它们分别采用粗号为8、16的画笔绘制线条，同时最多支持5条轨迹。

|  |  |
| --- | --- |
| 图 4.1 8号画笔 | 图 4.2 16号画笔 |

## 故障与调试

### 绘制线条间断

故障现象：在用单像素宽度线条进行测试时发现线条存在间断现象，但触摸事件调试信息给出的坐标点是连续的。

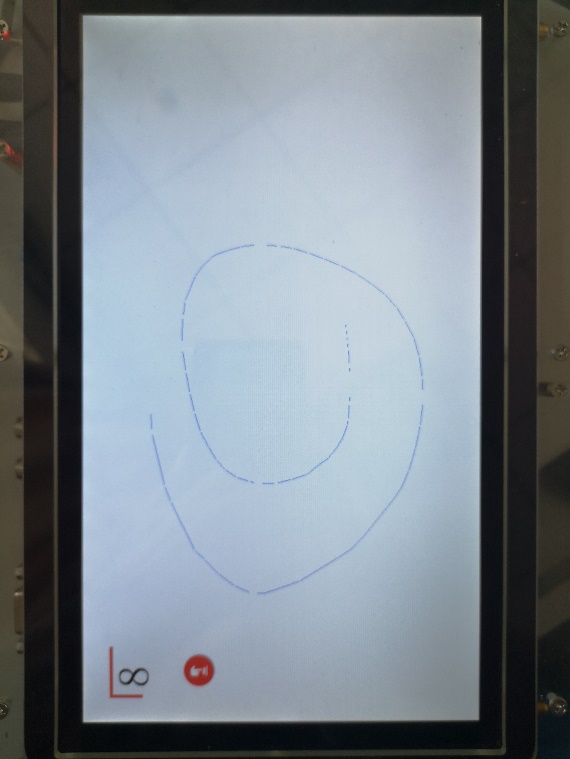


图 4.3 线条间断

原因分析：经过一轮筛查发现，线条间断问题与LCD写入速度无关，与画线算法也无关；该问题产生的原因是更新区域设置错误，出错代码如下。

|  |
| --- |
| int\* buf = \_begin\_draw(beg\_x, beg\_y, dx, dy); //长宽各缺少1个像素 |

解决方案：更新区域长宽各增加1个像素。

|  |
| --- |
| int\* buf = \_begin\_draw(beg\_x, beg\_y, dx + 1, dy + 1); |

# 蓝牙无线互联通讯

## 实验要求

本实验要求正确配置并启动蓝牙服务；熟练使用蓝牙的常用命令工具；通过RFCOMM协议（蓝牙串行通讯）实现无线通讯，通过蓝牙通讯测试。

## 方案设计

### 实验目标

本实验为验证实验，实验目的为正确配置并启动蓝牙服务；熟练使用蓝牙的常用命令工具；通过RFCOMM协议（蓝牙串行通讯）实现无线通讯，通过蓝牙通讯测试。实验目标为正确配置平台蓝牙模块，正确连接蓝牙，并运行测试程序进行测试。

### 蓝牙配置命令

在我们实验平台的中，蓝牙硬件是通过串口控制器和系统相连，Linux内核层中的硬件驱动和蓝牙协议层的相关模块都已经加载。在系统启动后，完成以下工作即可启动蓝牙服务：

1.设置蓝牙设备被发现时的名称：

/system/etc/bluetooth/main.conf 第12行：

Name = XXXX

2.设置蓝牙设备的mac地址：

/system/etc/bluetooth/bdaddr\_mac，

48bits，用6个字节来表示，例如12:34:56:78:9a:bc。

周围蓝牙设备的mac地址必须唯一。

3.开启串口蓝牙设备：

hciattach -n -s 115200 /dev/ttySAC3 rda 1152000 &

4.启动蓝牙服务程序 bluetoothd： setprop ctl.start bluetoothd

5.启动蓝牙设备配对程序： agent 1234 &

6.添加串口服务： sdptool add SP

7.连接设备

方法1： 一个实验平台做服务器，运行 rfcomm -r watch 0 1

另一个实验平台做客户端，运行 rfcomm -r connect 0 server\_mac 1

方法2： 一个实验平台做服务器，运行 hciconfig hci0 piscan rfcomm -r watch 0 1 手机上安装 “SPP蓝牙串口” 、 ”BluetoothSerial”或其他蓝 牙串口程序。 在手机上搜索实验平台的蓝牙设备，可能需要配对； 然后进入蓝牙串口程序，进行连接。

### 工具函数

蓝牙程序需要的工具函数如表5. 1所示。

表5. 1 蓝牙模块工具函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| myWrite\_nonblock | 经蓝牙设备发送相关信息。 |
| myRead\_nonblock | 经蓝牙设备读取相关信息。 |

## 结果验证

将蓝牙测试程序交叉编译，获得可执行文件lab5，将可执行文件上传到平台。在测试平台执行start\_bt.sh脚本文件，完成配置后，运行hciconfig hci0 piscan、rfcomm -r watch 0 1打开蓝牙连接。在手机上下载SPP蓝牙串口程序，找到蓝牙设备，连接蓝牙，app界面显示蓝牙已连接。在平台上运行lab5测试程序，在程序中向平台发送字符串"hello world!"，平台显示器上显示出收到的字符串信息；点击平台上的send按钮，app程序界面中显示接收到的hello字符串信息。如图5. 1所示。



图5. 1 lab5测试程序运行情况

# 综合设计——联机五子棋游戏

## 实验要求

综合运用所学内容，设计并实现一个在多个开发板之间进行蓝牙互联协作的程序或一个功能比较复杂的单机程序。

在本项设计内容中，我们将实现联机五子棋游戏。

## 方案设计

### 游戏需求分析

游戏需要使用平台提供的蓝牙和触摸屏幕，整个五子棋游戏程序应该使用事件进行驱动，在不同状态下响应不同的事件并执行不同操作。

为实现游戏逻辑，需要设计游戏状态，设置游戏状态转移图。为提高游戏体验，本五子棋游戏将全部使用图形界面进行操作，且两台主机为对等机，使用同一个可执行程序，均可选择棋子颜色，在游戏结束之后可通过重新开始按钮重新开始游戏。

### 游戏状态与状态转移图设计

为实现游戏逻辑，设计了如下游戏状态，如表6. 1所示。

表6. 1 游戏状态设计

|  |  |
| --- | --- |
| 状态辨识名 | 描述 |
| state\_0 | 等待游戏开始状态。 |
| state\_01 | 完成等待，等待对手机进行准备。 |
| state\_02 | 对手机完成准备，登台本机完成准备。 |
| state\_1 | 棋色选择阶段状态。 |
| state\_2 | 本机选中黑色棋子，等待对手机棋子颜色。 |
| state\_3 | 本机选中白色棋子，等待对手机选择棋子颜色。 |
| state\_4 | 对手机选中黑棋，等待本机选中棋子颜色（此时本机黑棋显示"已被选择"） |
| state\_5 | 对手机选中白棋，等待本机选择颜色（此时本机白棋显示"已选择"） |
| state\_6 | 本机执子等待落子 |
| state\_7 | 对手机执子等待落子 |
| state\_8 | 本机获胜，等待重新开始 |
| state\_9 | 对手机罗生，等待重新开始 |

游戏状态转移图如图6. 1所示。



图6. 1 状态转移图

### 蓝牙通信信息设置

程序中需要使用到蓝牙信息，这里设计一个结构体Message作为蓝牙通信的信息单元。Message蓝牙信息单元定义如下。

|  |
| --- |
| /\*蓝牙通信结构体\*/  typedef struct Message{  int type; /\*消息类型\*/  int x;/\*落子坐标x轴\*/  int y;/\*落子坐标y轴\*/  }Message; |

本游戏程序中，将蓝牙通信信息类型分为以下五类，如表6. 2所示。

表6. 2 蓝牙通信消息类型

|  |  |
| --- | --- |
| 消息类型编号 | 描述 |
| 1 | 表示“准备完成”。此时坐标信息字段无效。 |
| 2 | 表示“选中白棋”。此时坐标信息字段无效。 |
| 3 | 表示“选中黑棋”。此时坐标信息字段无效。 |
| 4 | 表示落下棋子，此时坐标信息字段表示落子坐标。 |
| 5 | 表示落下棋子且本机获胜，此时坐标信息表示落子坐标。 |

将蓝牙信息发送动作封装成函数，如下所示。

|  |
| --- |
| /\*蓝牙信息发送\*/  void bluetooth\_send(int ctype,int cx,int cy,int fd){  Message \*message = (Message\*)malloc(sizeof(Message));  message->type=ctype;  message->x=cx;  message->y=cy;  myWrite\_nonblock(fd, message, sizeof(Message));  printf("send message:type:%d,size:%d\n",message->type,sizeof(Message));  free(message);  } |

### 图形界面按钮组件封装

为实现图形界面按钮的添加和点击事件处理，将按钮组件进行封装。按钮结构体定义如下。

|  |
| --- |
| /\*按钮事件结构体与按钮事件响应函数指针\*/  typedef struct ButtonEvent{  int enable; //使能，为1时表示可用，为0时表示禁止使用  //坐标信息  int x\_min;  int y\_min;  int x\_max;  int y\_max;  int (\*p)(struct ButtonEvent\* button); //函数指针，绑定的按钮响应函数  }ButtonEvent; |

按钮队列，编写函数检测该队列，并执行绑定的按钮响应函数。按钮队列结构体如下。

|  |
| --- |
| /\*按钮队列，最多仅允许存在10个按钮\*/  typedef struct ButtonQueue{  int tail;  ButtonEvent \*buttons[10];  }ButtonQueue;  /\*屏幕中存在的按钮队列\*/  ButtonQueue buttonList; /\*全局变量，按钮队列\*/ |

对按钮的添加、删除、清空按钮队列进行封装，封装的相关函数如下。

|  |
| --- |
| /\*清空按钮队列\*/  void clearButtonList(){  int i = 0;  for(i = 0;i < 10;i ++){  if(buttonList.buttons[i]!=NULL){  free(buttonList.buttons[i]);  buttonList.buttons[i] = NULL;  }  }  buttonList.tail=0;  }  /\*添加按钮到队尾\*/  int addButtonToList(ButtonEvent \* button){  if(buttonList.tail==10) return -1;//添加失败  buttonList.buttons[buttonList.tail++]=button;  return 0;  }  /\*按钮生成和显示\*/  int initButton(int x,int y,int w,int h,int color,char str[],int fontSize,int (\*fp)(struct ButtonEvent\* button)){  //添加按钮  ButtonEvent\* button = (ButtonEvent\*)malloc(sizeof(ButtonEvent));  button->x\_min=x;  button->y\_min=y;  button->x\_max=x+w;  button->y\_max=y+h;  button->p=fp;  button->enable = 1;  //绘制按钮信息  fb\_draw\_rect(x,y,w,h,COLOR\_BACKGROUND);  fb\_draw\_border(x,y,w,h,color);  //绘制字符串到正中心  //获得字符串长度  int length=0;  while(str[length]!='\0') length++;  printf("length:%d",length);  int text\_x = x + (w - length \* fontSize/3)/2;  int text\_y = y + (h + fontSize)/2;  //绘制字体  fb\_draw\_text(text\_x,text\_y,str,fontSize,color);  //添加到按钮队列中  addButtonToList(button);  fb\_update();  return 0;  } |

在程序中需要对按钮队列进行检测，检测和触发按钮响应事件的函数如下。

|  |
| --- |
| /\*按钮触发检测，调用触发的函数\*/  void buttonListClicked(int x,int y){  int i = 0;  printf("buttonList:\n");  for(i = 0;i < buttonList.tail;i++){  printf("button%d:x\_min:%d,x\_max:%d,y\_min:%d,y\_max:%d,enable:%d\n",i,buttonList.buttons[i]->x\_min,buttonList.buttons[i]->x\_max,buttonList.buttons[i]->y\_min,buttonList.buttons[i]->y\_max,buttonList.buttons[i]->enable);  }  printf("button check\n");  for(i = buttonList.tail-1;i>=0;i--){  printf("i=%d\n",i);  if(buttonList.buttons[i]->enable==0) continue;  if(x<buttonList.buttons[i]->x\_max&&x>buttonList.buttons[i]->x\_min&&y<buttonList.buttons[i]->y\_max&&y>buttonList.buttons[i]->y\_min){  /\*操作顺序：添加按钮--检测按钮--根据返回值调用按钮响应\*/  if(buttonList.buttons[i]->p==NULL) continue;  (\*(buttonList.buttons[i]->p))(buttonList.buttons[i]);  break; /\*仅检测最顶层按钮，执行相关响应事件后跳出\*/  }  }  printf("button check end\n");  } |

### 五子棋逻辑实现

将五子棋的基本逻辑需要的函数进行封装，封装的函数如表6. 3所示。

表6. 3 游戏逻辑处理封装函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| initGame | 无参函数，初始化游戏。重置游戏棋盘，初始化游戏状态，清空按钮队列。 |
| drawChessBoard | 用灰色背景覆盖背景，绘制棋盘线条。 |
| initChessboard | 绘制棋盘线条。 |
| cauculateCircleFunc | 计算圆型函数x^2+y^2=17^2的横纵坐标对应相关值，储存到全局数组中，用于绘制圆形棋子。 |
| drawPiece | 在棋盘中绘制棋子。 |
| checkBoard | 落子后检测棋盘，若恰好5颗棋子连成一线则返回颜色信息，否则返回0。 |

### 按钮响应事件

游戏逻辑需要的按钮响应事件定义如所示。

表6. 4 按钮响应事件

|  |  |
| --- | --- |
| 函数名 | 描述 |
| failureButtonEvent | 游戏失败的确认按钮，点击后将重新开始游戏。 |
| startButtonHandler | 开始按钮响应事件。 |
| selectBlack | 选择黑色棋子按钮响应事件。 |
| selectWhite | 选择白色棋子按钮响应事件。 |

### 游戏状态转换实现

在蓝牙信息处理模块和触摸按键处理逻辑中对游戏状态的转移和事件处理进行处理。如下所示。

|  |
| --- |
| /\*蓝牙信息接收模块\*/  void bluetooth\_tty\_event\_cb(int fd)  {  Message message;  int n;  n = myRead\_nonblock(fd, &message, sizeof(Message));  if(n <= 0) {  printf("close bluetooth tty fd\n");  //task\_delete\_file(fd);  //close(fd);  exit(0);  return;  }  //接收信息  printf("get message:type:%d\n",message.type);  switch(message.type){    case 1:  switch(game\_state){  case state\_0: /\*状态0 接收信息\*/  /\*对手完成准备，本机状态改变\*/  printf("state\_0 to state\_02\n");  game\_state = state\_02;  break;  case state\_01:  /\*完成准备，绘制选择界面\*/  printf("state\_01 to state\_1\n");  game\_state = state\_1;  drawSelect();  break;  default:  error("游戏状态转换，发送消息接收信息type1");  }  break;  case 2:  switch(game\_state){  case state\_1:  printf("state\_1 to state\_5\n");  game\_state = state\_5;  /\*白棋被选择\*/  /\*设置被选择信息，并将白棋选择按钮使能关闭\*/  buttonList.buttons[1]->enable=0;  initButton(512,200,200,200,WHITE,"已被选择",30,NULL);  fb\_update();  break;  case state\_2:  /\*清空按钮队列，绘制棋盘，转入状态6\*/  printf("state\_2 to state\_6\n");  game\_state = state\_6;  clearButtonList();  drawChessBoard();  break;  default:  error("游戏状态转换，接收消息type2");  }  break;  case 3:  switch(game\_state){  case state\_1:  printf("state\_1 to state\_4\n");  game\_state = state\_4;  /\*黑棋被选择\*/  /\*设置被选择信息，并将黑棋选择按钮使能关闭\*/  buttonList.buttons[0]->enable=0;  initButton(312,200,200,200,BLACK,"已被选择",30,NULL);  fb\_update();  break;    case state\_3:  /\*清空按钮队列，绘制棋盘，转入状态7\*/  printf("state\_3 to state\_7\n");  game\_state = state\_7;  clearButtonList();  drawChessBoard();  break;    default:  error("游戏状态转换，接收消息type3");  }  break;  case 4:  switch(game\_state){  case state\_7:  printf("state\_7 to state\_6\n");  game\_state = state\_6;  /\*分析相关信息\*/  /\*绘制棋子\*/  drawPiece(message.x,message.y,dis\_draw\_color); //队首棋颜色绘制    break;  default:  error("游戏状态转换，接收消息type4");  }  break;  case 5:  switch(game\_state){  case state\_7:  printf("state\_7 to state\_9\n");  game\_state = state\_9;  //打印本机失败并创建重新开始的按钮  initButton(312,200,400,200,RED,"失败",30,failureButtonEvent);  fb\_update();  break;  default:  error("游戏状态转换，接收消息type5");  }  break;  default:  error("游戏状态转换，消息类型不正确");  }  return;  }  /\*触摸板事件控制模块，根据触摸位置和状态信息作出反应\*/  void touch\_event\_cb(int fd){  int type,x,y,finger;  int no\_x,no\_y;  type = touch\_read(fd, &x,&y,&finger);  switch(type){  case TOUCH\_PRESS:  //按钮检测  printf("%d,%d\n",x,y);  printf("game\_state=%d\n",game\_state);  if(game\_state == state\_6){  printf("game\_color=%d\n",game\_color);  /\*获得坐标位置信息，发送信息\*/  no\_x = (x-(242-18))/36;  no\_y = (y-(30-18))/36;  if(no\_x<0||no\_x>15||no\_y<0||no\_y>15) break;  if(board[no\_x][no\_y]!=0) break;  /\*绘制\*/  drawPiece(no\_x,no\_y,draw\_color);  fb\_update();  if(checkBoard(no\_x,no\_y)!=0){  /\*发送信息，状态修改\*/  bluetooth\_send(5,no\_x,no\_y,bluetooth\_fd);  game\_state = state\_8;  //添加按钮，本机获胜  //打印本机失败并创建重新开始的按钮  initButton(312,200,400,200,GREEN,"获胜",30,failureButtonEvent);  fb\_update();  }else{  /\*状态修改\*/  game\_state = state\_7;    /\*发送坐标信息\*/  bluetooth\_send(4,no\_x,no\_y,bluetooth\_fd);  }  }else if(game\_state == state\_7){}  else{  printf("checked\n");  buttonListClicked(x,y);  }  break;  case TOUCH\_ERROR:  printf("close touch fd\n");  break;  default:  return;  }  } |

## 结果验证

交叉编译，生成可执行文件gobang，将可执行文件上传到两个平台设备上，连接两台蓝牙设备。在两台设备上运行gobang程序。游戏状态显示游戏开始界面，如图6. 2所示。

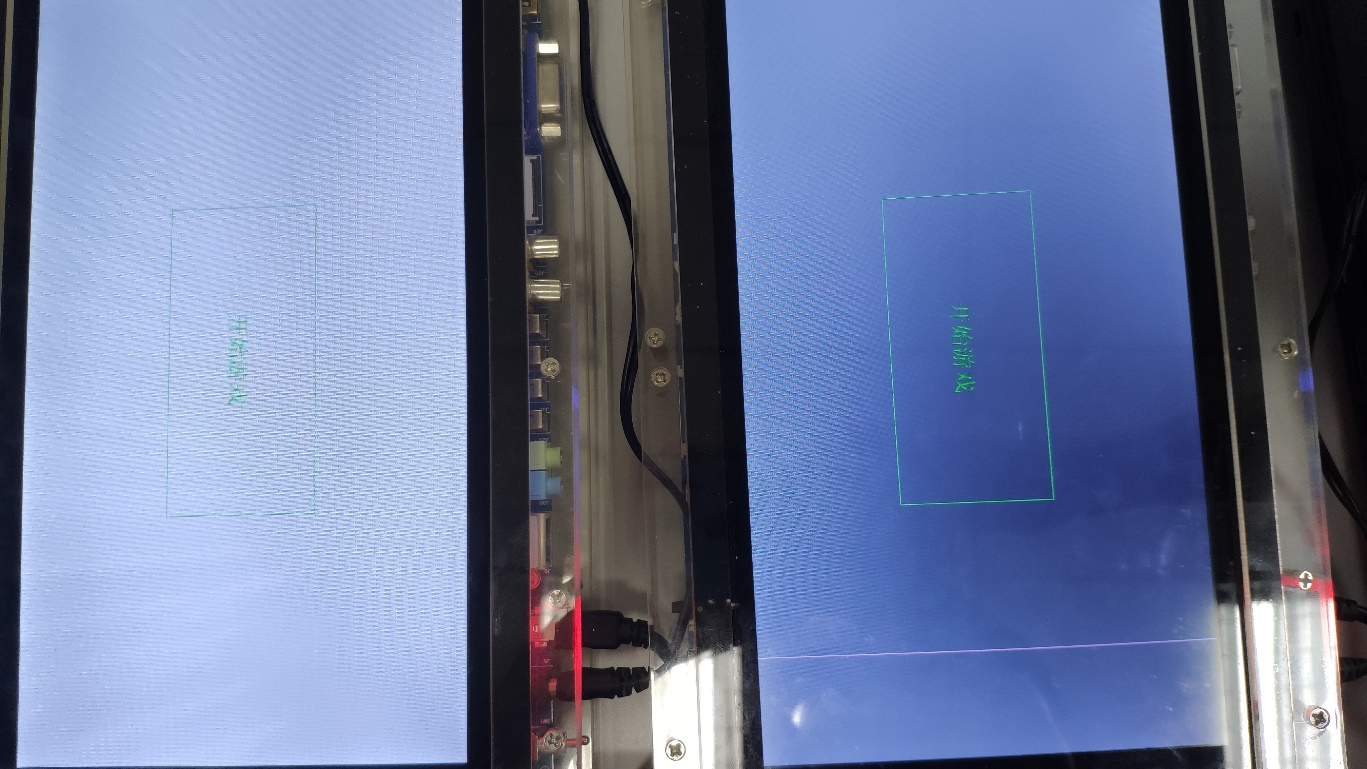


图6. 2 游戏开始界面

一台设备点击开始游戏按钮后，屏幕清空，当另一台点击开始游戏按钮后，两台设备均进入棋子选择阶段，如图6. 3所示。

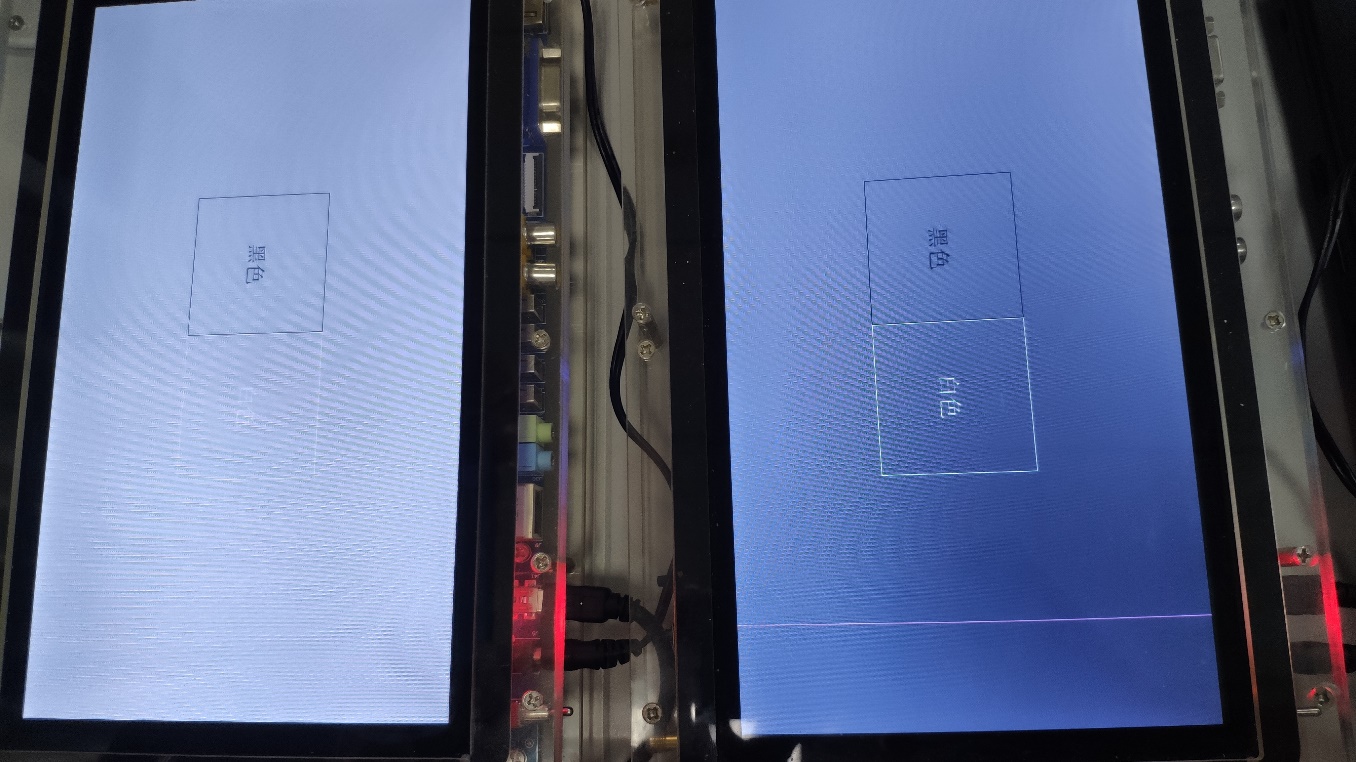


图6. 3 棋子选择界面

其中一台机器选中黑棋，选中黑棋的设备显示黑棋已选择，另一台显示黑色已被选择。如图6. 4所示。

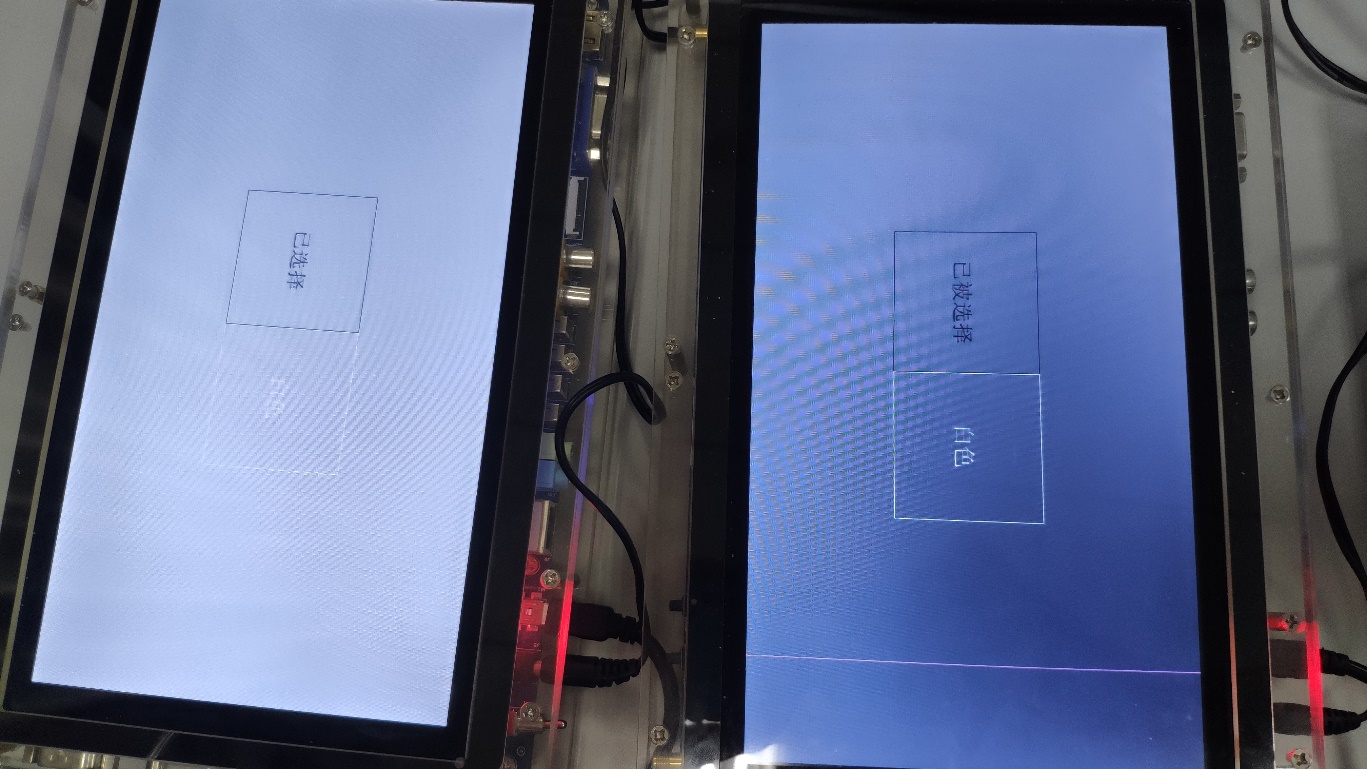


图6. 4 黑棋已选中界面

另一台机器选中白色棋子，游戏进入落子阶段，屏幕中显示棋盘界面。如图6. 5所示。

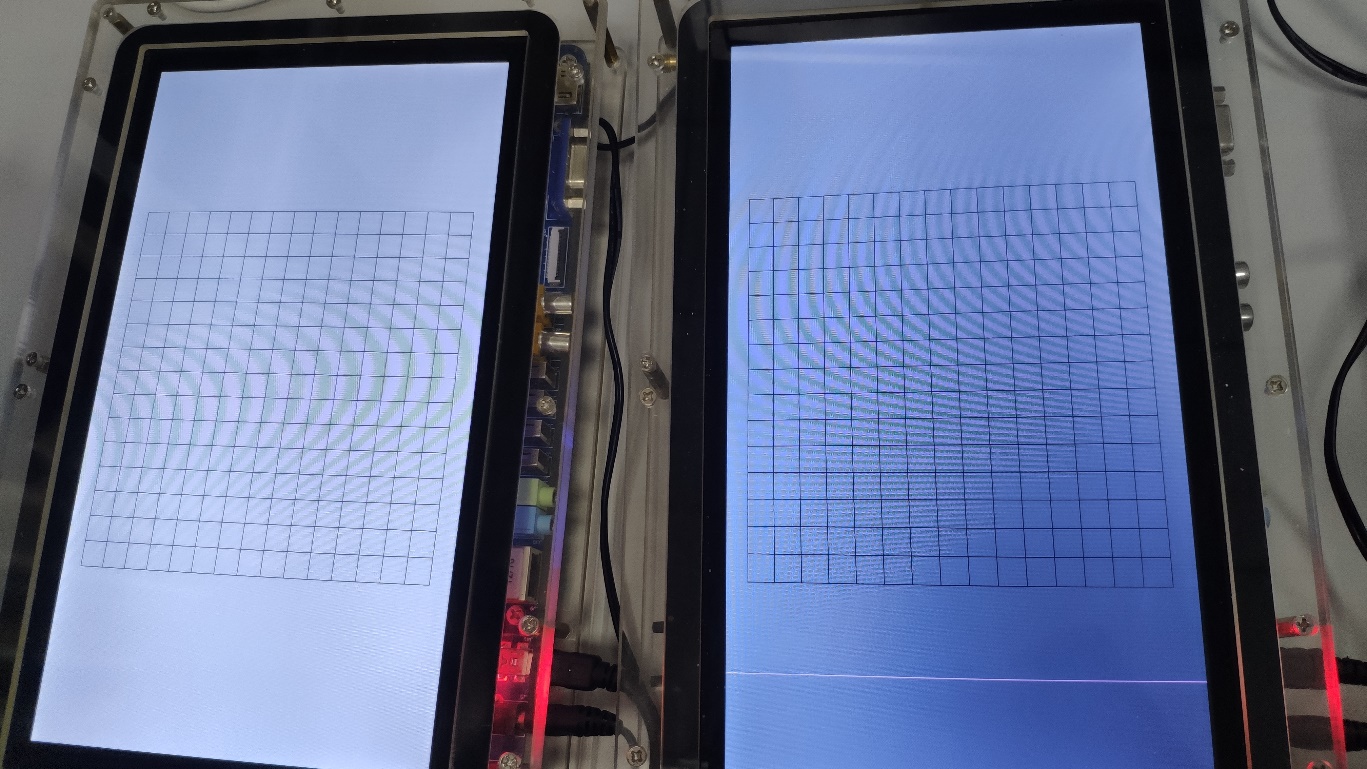


图6. 5 开始下棋界面

在其中一台机器上落子，另一台设备屏幕上同步出现落子信息，双方轮流落子，如图6. 6所示。

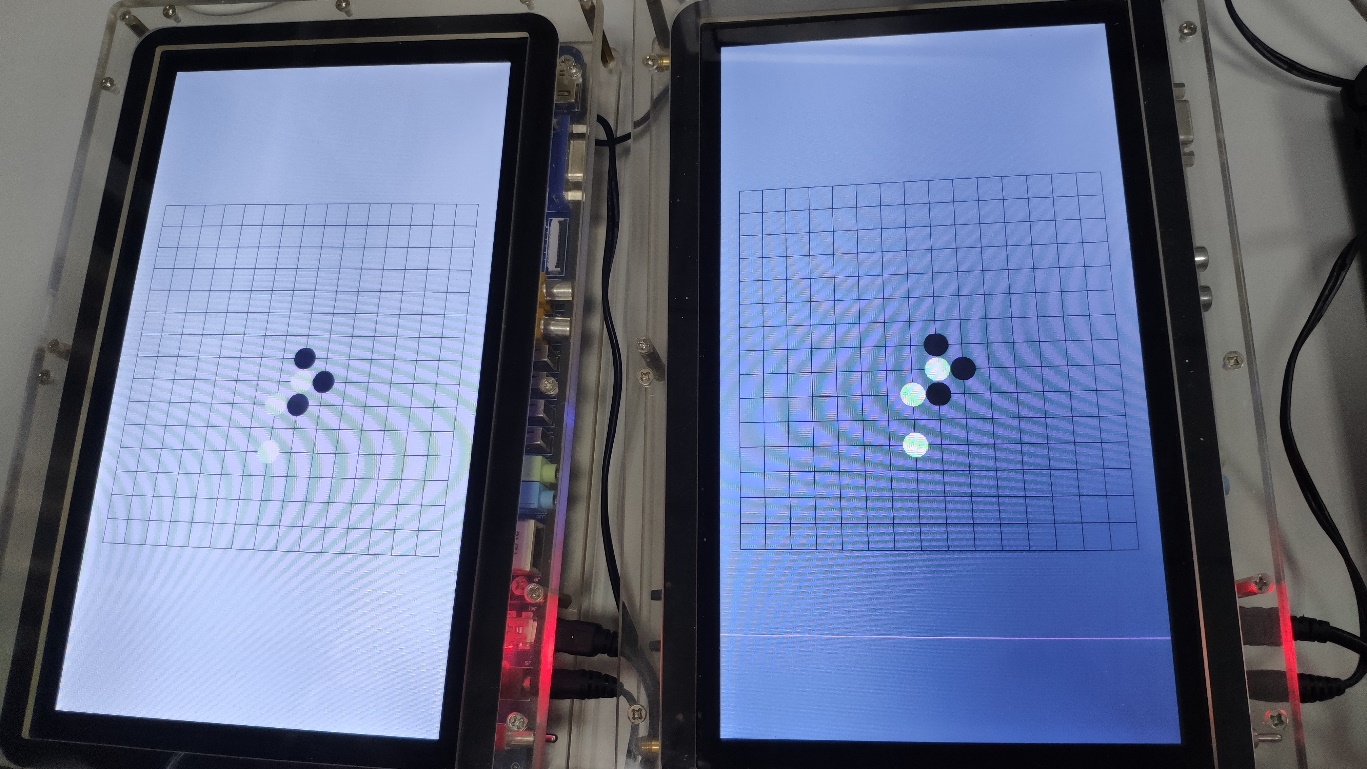


图6. 6 轮流落子界面

当其中一种颜色棋子获胜，获胜方显示胜利信息，另一方显示失败信息，如图6. 7所示。

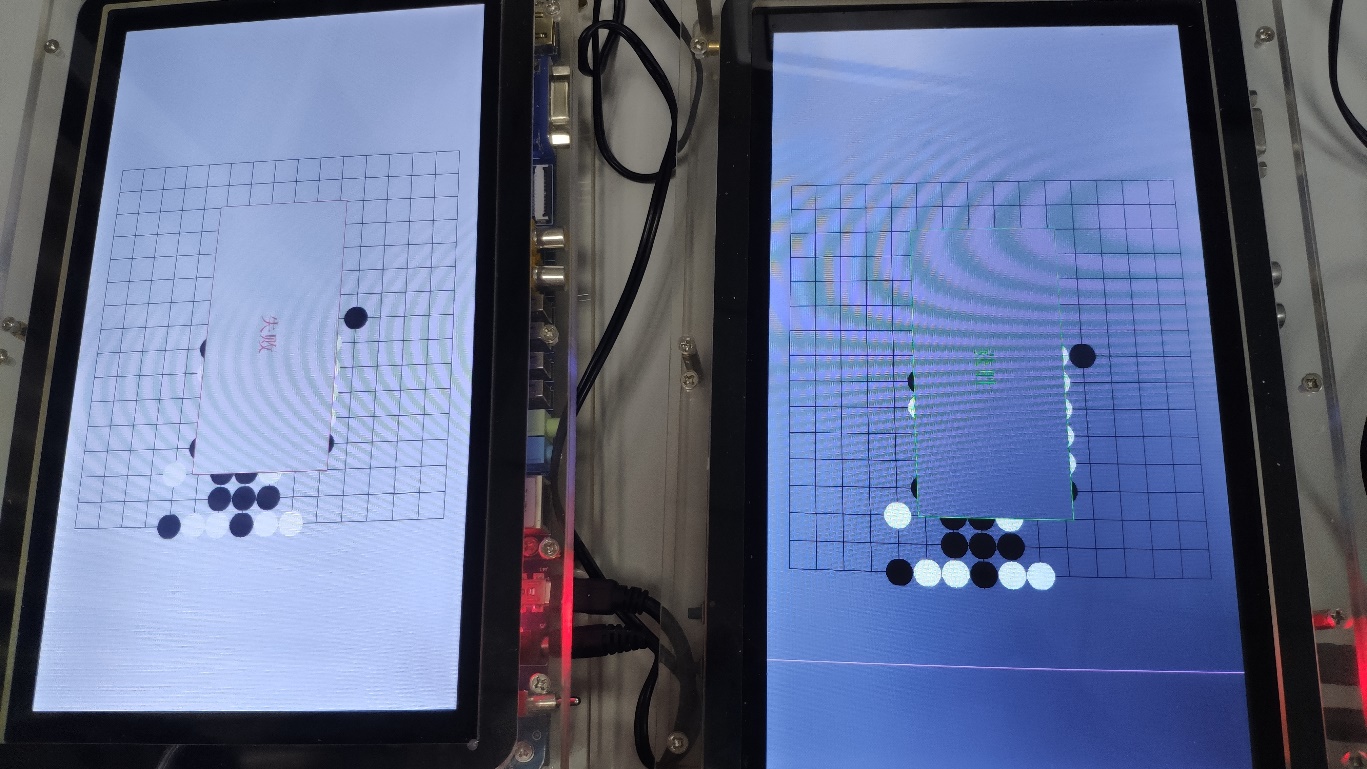


图6. 7 游戏结束界面

两台设备分别点击按钮，游戏程序转到重新开始界面。此时可开始下一次游戏。

## 故障与调试

### 段溢出

**故障现象：**在游戏运行过程中，在选中按钮后，总出现段溢出的情况，程序终止运行。

**原因分析：**在按钮事件检测可触发函数中，原编写的函数如下。

|  |
| --- |
| for(i = buttonList.tail-1;i>=0;i--){  printf("i=%d\n",i);  if(buttonList.buttons[i]->enable==0) continue;  if(x<buttonList.buttons[i]->x\_max&&x>buttonList.buttons[i]->x\_min&&y<buttonList.buttons[i]->y\_max&&y>buttonList.buttons[i]->y\_min){  /\*操作顺序：添加按钮--检测按钮--根据返回值调用按钮响应\*/  if(buttonList.buttons[i]->p==NULL) continue;  (\*(buttonList.buttons[i]->p))(buttonList.buttons[i]);  }  } |

而在棋子选择按钮响应事件中，将按钮队列清空，故而在下一次循环中访问buttonList.buttons[i]时出现空引用的状况，导致程序不正常退出。

**解决方案：**设置按钮响应仅响应最顶层按钮信息，在处理一个按钮事件后，跳出事件检测。修改后的检测函数如下所示。

|  |
| --- |
| for(i = buttonList.tail-1;i>=0;i--){  printf("i=%d\n",i);  if(buttonList.buttons[i]->enable==0) continue;  if(x<buttonList.buttons[i]->x\_max&&x>buttonList.buttons[i]->x\_min&&y<buttonList.buttons[i]->y\_max&&y>buttonList.buttons[i]->y\_min){  /\*操作顺序：添加按钮--检测按钮--根据返回值调用按钮响应\*/  if(buttonList.buttons[i]->p==NULL) continue;  (\*(buttonList.buttons[i]->p))(buttonList.buttons[i]);  break; /\*仅响应最顶层按钮，完成响应事件后退出\*/  }  } |

### 棋子颜色反转

**故障现象**：在落子后，若在已经落下地方对手棋子的地方点击，则会将对手的棋子颜色翻转为自己的棋子。

**故障分析**：落子时，未对棋盘已有棋子数据进行检测。

**修改方案**：在落子之前，对已有棋盘进行检测，若已存在棋子，则不允许落子。

# 总结与心得

## 实验总结

本次实验主要完成了如下几点工作：

1）完成linux内核编译和系统烧录。

2）完成矩形、线条绘制封装函数的编写。

3）完成图片显示和文本显示封装函数的编写。

4）多点触控开发。

5）蓝牙无线互联函数

6）设计、编写并测试联机五子棋游戏。

## 实验心得

本次实验课程，从简入繁，从编译和烧录内核开始，一步步深入，利用接口函数使用实验平台的显示屏、触摸屏和蓝牙模块，编写实现图形库函数。最后综合学习到的知识，设计和实现了联机五子棋游戏。

此次实验，是对理论课程的巩固和深化，让我们在一边学习一边实现的过程中体会到了利用实验平台进行实验的乐趣。实验难度较为适中，在实验课程提供的框架下编写图形库封装函数和调通实现触摸屏以及蓝牙模块的使用示例代码，我们学习并理解了平台相关模块的使用方法。最后的综合实验能够很大激发我们的创造力，利用前面的实验中学到的知识设计自己的联机程序。课程一步步深入，循序渐进，在此次实验课程中，我收获颇丰。不光学会了屏幕显示信息处理函数的编写方法，学会了触摸事件响应处理，学会了蓝牙模块的信息发送和接收，还进一步锻炼了自己的动手能力。

在此次实验中，遇到了许多问题，比如在编写图片显示封装函数时，由于没有考虑到图片宽度与屏幕可用宽度的问题，虽然测试程序并不能完全展示出问题，但自己编写的测试函数并不能正常显示，最后通过修改终于正确完成了实验。在进行综合实验的过程中，设计和初步编写过程中均一帆风顺，但最后的调试阶段花费了我们很长时间，总是会出现段错误的情况，最后我们一步步调试，最后终于解决了问题，完成了联机五子棋程序的编写。

# 附录

## 图形库代码

图形库处理代码文件graphic.c如下。

|  |
| --- |
| #include "common.h"  #include <sys/ioctl.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\_FB\_FD;  static int \*LCD\_FB\_BUF = NULL;  static int \*LCD\_FB\_FRONT, \*LCD\_FB\_BACK;  struct fb\_var\_screeninfo LCD\_FB\_VAR;  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)  void fb\_init(char \*dev)  {  int fd;  struct fb\_fix\_screeninfo fb\_fix;  struct fb\_var\_screeninfo fb\_var;  if(LCD\_FB\_BUF != 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;  }  printf("framebuffer info: bits\_per\_pixel=%u,size=(%d,%d),virtual\_pos\_size=(%d,%d)(%d,%d),line\_length=%u,smem\_len=%u\n",  fb\_var.bits\_per\_pixel, fb\_var.xres, fb\_var.yres, fb\_var.xoffset, fb\_var.yoffset,  fb\_var.xres\_virtual, fb\_var.yres\_virtual, fb\_fix.line\_length, fb\_fix.smem\_len);  //Second: mmap  int \*addr;  addr = mmap(NULL, fb\_fix.smem\_len, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0);  if((int)addr == -1){  printf("failed to mmap memory for framebuffer.\n");  return;  }  if((fb\_var.xoffset != 0) ||(fb\_var.yoffset != 0))  {  fb\_var.xoffset = 0;  fb\_var.yoffset = 0;  if(ioctl(fd, FBIOPAN\_DISPLAY, &fb\_var) < 0) {  printf("FBIOPAN\_DISPLAY framebuffer failed\n");  }  }  LCD\_FB\_FD = fd;  LCD\_FB\_BUF = addr;  LCD\_FB\_FRONT = addr;  LCD\_FB\_BACK = addr + fb\_var.xres\*fb\_var.yres;  LCD\_FB\_VAR = fb\_var;  //set empty  AREA\_SET\_EMPTY(&update\_area);  return;  }  static void \_copy\_area(int \*dst, int \*src, struct area \*pa)  {  int x, y, w, h;  x = pa->x1; w = pa->x2-x;  y = pa->y1; h = pa->y2-y;  src += y\*SCREEN\_WIDTH + x;  dst += y\*SCREEN\_WIDTH + x;  while(h-- > 0){  memcpy(dst, src, w\*4);  src += SCREEN\_WIDTH;  dst += SCREEN\_WIDTH;  }  }  static int \_check\_area(struct area \*pa)  {  if(pa->x2 == 0) return 0; //is empty  if(pa->x1 < 0) pa->x1 = 0;  if(pa->x2 > SCREEN\_WIDTH) pa->x2 = SCREEN\_WIDTH;  if(pa->y1 < 0) pa->y1 = 0;  if(pa->y2 > SCREEN\_HEIGHT) pa->y2 = SCREEN\_HEIGHT;  if((pa->x2 > pa->x1) && (pa->y2 > pa->y1))  return 1; //no empty  //set empty  AREA\_SET\_EMPTY(pa);  return 0;  }  void fb\_update(void)  {  if(\_check\_area(&update\_area) == 0) return; //is empty  \_copy\_area(LCD\_FB\_FRONT, DRAW\_BUF, &update\_area);  AREA\_SET\_EMPTY(&update\_area); //set empty  return;  }  /\*======================================================================\*/  /\*refresh update\_area\*/  static void \* \_begin\_draw(int x, int y, int w, int h)  {  int x2 = x+w;  int y2 = y+h;  if(update\_area.x1 > x) update\_area.x1 = x;  if(update\_area.y1 > y) update\_area.y1 = y;  if(update\_area.x2 < x2) update\_area.x2 = x2;  if(update\_area.y2 < y2) update\_area.y2 = y2;  return DRAW\_BUF;  }  void fb\_draw\_pixel(int x, int y, int color)  {  if(x<0 || y<0 || x>=SCREEN\_WIDTH || y>=SCREEN\_HEIGHT) return;  int \*buf = \_begin\_draw(x,y,1,1);  /\*---------------------------------------------------\*/  \*(buf + y\*SCREEN\_WIDTH + x) = color;  /\*---------------------------------------------------\*/  return;  }  void fb\_draw\_rect(int x, int y, int w, int h, int color)  {  if(x < 0) { w += x; x = 0;}  if(x+w > SCREEN\_WIDTH) { w = SCREEN\_WIDTH-x;}  if(y < 0) { h += y; y = 0;}  if(y+h >SCREEN\_HEIGHT) { h = SCREEN\_HEIGHT-y;}  if(w<=0 || h<=0) return;  int \*buf = \_begin\_draw(x,y,w,h);  /\*---------------------------------------------------\*/  printf("you need implement fb\_draw\_rect()\n"); exit(0);  /\*---------------------------------------------------\*/  return;  }  void fb\_draw\_line(int x1, int y1, int x2, int y2, int color)  {  /\*---------------------------------------------------\*/  printf("you need implement fb\_draw\_line()\n"); exit(0);  /\*---------------------------------------------------\*/  return;  }  void fb\_draw\_image(int x, int y, fb\_image \*image, int color)  {  if(image == NULL) return;  int ix = 0; //image x  int iy = 0; //image y  int w = image->pixel\_w; //draw width  int h = image->pixel\_h; //draw height  if(x<0) {w+=x; ix-=x; x=0;}  if(y<0) {h+=y; iy-=y; y=0;}    if(x+w > SCREEN\_WIDTH) {  w = SCREEN\_WIDTH - x;  }  if(y+h > SCREEN\_HEIGHT) {  h = SCREEN\_HEIGHT - y;  }  if((w <= 0)||(h <= 0)) return;  int \*buf = \_begin\_draw(x,y,w,h);  /\*---------------------------------------------------------------\*/  char \*dst = (char \*)(buf + y\*SCREEN\_WIDTH + x);  char \*src = image->content + iy\*image->line\_byte + ix\*4;  /\*---------------------------------------------------------------\*/  int alpha;  int R1,G1,B1;  if(image->color\_type == FB\_COLOR\_RGB\_8880) /\*lab3: jpg\*/  {  /\*get a line of colors and copy them to buffer\*/  int length = image->line\_byte;  if(image->line\_byte > 4 \* w) length = w\*4; /\*close to the border\*/  for(int i=0;i<h;i++){  memcpy(dst,src,length);  dst += SCREEN\_WIDTH; //shift  src += image->line\_byte; //shift  }  return;  }  if(image->color\_type == FB\_COLOR\_RGBA\_8888) /\*lab3: png\*/  {  for(int i = 0;i < w,i++){  for(int j = 0;j < h;j++){  //当前点的显存地址信息  int \* addr = buf + (i+x)\*SCREEN\_WIDTH + y + j;  char \*p = (char\*)addr;  //获得当前点的RGB样式  int argb = \*addr;  //图像信息  int pic\_argb = \*(image->content + i\*image->line\_byte/4 + j);  alpha = pic\_rgba >> 24;  R1 = (argb & 0xff0000)>>16;  G1 = (argb & 0xff00)>>8;  R1 = argb & 0xff;  switch(alpha){  case 0:break;  case 255:\*(addr) = pic\_argb;break;  default:  p[0] += (((B1 - p[0]) \* alpha) >> 8);  p[1] += (((G1 - p[1]) \* alpha) >> 8);  p[2] += (((R1 - p[2]) \* alpha) >> 8);  }  }  }  return;  }  if(image->color\_type == FB\_COLOR\_ALPHA\_8) /\*lab3: font\*/  {  R1 = (color&0xff0000)>>16;  G1 = (color&0xff00)>>8;  B1 = color&0xff;  //字模  for(int i = 0;i < h;i++){  for(int j = 0;j < w;j++){  int\* addr = buf + (i + x) \* SCREEN\_WIDTH + y + j;  char\* p = (char\*)addr;  //获得Alpha  alpha = \*(char\*)(image->content + i \* image->line\_byte + j);  switch (alpha) {  case 0:break;  case 255:\*(addr) = pic\_argb; break;  default:  p[0] += (((B1 - p[0]) \* alpha) >> 8);  p[1] += (((G1 - p[1]) \* alpha) >> 8);  p[2] += (((R1 - p[2]) \* alpha) >> 8);  }  }  }  return;  }  /\*---------------------------------------------------------------\*/  return;  }  void fb\_draw\_border(int x, int y, int w, int h, int color)  {  if(w<=0 || h<=0) return;  fb\_draw\_rect(x, y, w, 1, color);  if(h > 1) {  fb\_draw\_rect(x, y+h-1, w, 1, color);  fb\_draw\_rect(x, y+1, 1, h-2, color);  if(w > 1) fb\_draw\_rect(x+w-1, y+1, 1, h-2, color);  }  }  /\*\* 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;  } |

## 联机五子棋游戏代码

游戏代码分为两个文件，gobang.h和main.c两个文件，如下所示。

(1)gobang.h

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <string.h>  #include "../common/common.h"  /\*颜色信息定义\*/  #define COLOR\_BACKGROUND FB\_COLOR(0xaf,0xaf,0xaf)  #define COLOR\_TEXT FB\_COLOR(0x0,0x0,0x0)  #define COLOR\_LINE FB\_COLOR(0x0,0x0,0x0)  #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)  /\*\*  \* 五子棋联机通信游戏头文件支持  \*/  /\*硬件信息\*/  static int touch\_fd;  static int bluetooth\_fd;  /\*游戏状态\*/  enum State{  state\_0, /\*开始阶段\*/  state\_01, /\*完成准备等待对手机准备\*/  state\_02, /\*对手机完成准备，等待本机准备\*/  state\_1, /\*棋色选择\*/  state\_2,  state\_3,  state\_4,  state\_5,  state\_6,  state\_7,  state\_8,  state\_9  };  /\*棋盘\*/  // 0--none 1--black 2--write  extern int board[][16];  /\*绘制函数\*/  extern int circleFunc[];  /\*游戏状态\*/  extern enum State game\_state;  /\*本机棋子颜色 : 1 - black ;2 - white \*/  extern int game\_color;  extern int draw\_color;  extern int dis\_draw\_color;//对手棋子颜色  /\*初始化函数：蓝牙设备初始化\*/  static int bluetooth\_tty\_init(const char \*dev)  {  int fd = open(dev, O\_RDWR|O\_NOCTTY|O\_NONBLOCK); /\*非阻塞模式\*/  if(fd < 0){  printf("bluetooth\_tty\_init open %s error(%d): %s\n", dev, errno, strerror(errno));  return -1;  }  return fd;  }  /\*按钮事件结构体与按钮事件响应函数指针\*/  typedef struct ButtonEvent{  int enable; //使能  //坐标信息  int x\_min;  int y\_min;  int x\_max;  int y\_max;  int (\*p)(struct ButtonEvent\* button); //函数指针  }ButtonEvent;  /\*按钮队列，最多仅允许存在10个按钮\*/  typedef struct ButtonQueue{  int tail;  ButtonEvent \*buttons[10];  }ButtonQueue;  /\*屏幕中存在的按钮队列\*/  extern ButtonQueue buttonList;  /\*蓝牙通信结构体\*/  typedef struct Message{  int type;  int x;  int y;  }Message;  void clearButtonList();  int addButtonToList(ButtonEvent \* button);  int initButton(int x,int y,int w,int h,int color,char str[],int fontSize,int (\*fp)(struct ButtonEvent\* button));  void buttonListClicked(int x,int y);  void initGame();  void initScreen();  void drawChessBoard();  void initChessboard();  void cauculateCircleFunc();  void drawPiece(int x,int y,int color);  int checkBoard(int x,int y);  void touch\_event\_cb(int fd);  void bluetooth\_tty\_event\_cb(int fd);  void bluetooth\_send(int ctype,int cx,int cy,int fd);  int failureButtonEvent(ButtonEvent\* button);  int startButtonHandler(ButtonEvent\* button);  void drawSelect();  int selectBlack(ButtonEvent\* button);  int selectWhite(ButtonEvent\* button);  void error(char str[]); |

(2)mian.c

|  |
| --- |
| #include "gobang.h"  ButtonQueue buttonList;  /\*棋盘\*/  // 0--none 1--black 2--write  int board[16][16];  /\*绘制函数\*/  int circleFunc[18];  /\*游戏状态\*/  enum State game\_state;  /\*本机棋子颜色 : 1 - black ;2 - white \*/  int game\_color;  int draw\_color;  int dis\_draw\_color;//对手棋子颜色  int main(int argc,char\*\*argv){  //初始化设备  printf("debug0\n");  fb\_init("/dev/graphics/fb0");  font\_init("/data/local/font.ttc");  printf("debug1\n");  touch\_fd = touch\_init("/dev/input/event3");  task\_add\_file(touch\_fd, touch\_event\_cb);  printf("init\n");  bluetooth\_fd = bluetooth\_tty\_init("/dev/rfcomm0");  if(bluetooth\_fd == -1) return 0;  task\_add\_file(bluetooth\_fd, bluetooth\_tty\_event\_cb);  //开始游戏，绘制开始界面，状态0  initGame();  //list init  memset(&buttonList,0,sizeof(ButtonQueue));  //绘制准备界面,添加准备按钮  initButton(312,200,400,200,GREEN,"开始游戏",30,startButtonHandler);    printf("debug2\n");  task\_loop();  return 0;  }  /\*清空按钮队列\*/  void clearButtonList(){  int i = 0;  for(i = 0;i < 10;i ++){  if(buttonList.buttons[i]!=NULL){  free(buttonList.buttons[i]);  buttonList.buttons[i] = NULL;  }  }  buttonList.tail=0;  }  /\*添加按钮到队尾\*/  int addButtonToList(ButtonEvent \* button){  if(buttonList.tail==10) return -1;//添加失败  buttonList.buttons[buttonList.tail++]=button;  return 0;  }  /\*按钮生成\*/  int initButton(int x,int y,int w,int h,int color,char str[],int fontSize,int (\*fp)(struct ButtonEvent\* button)){  //添加按钮  ButtonEvent\* button = (ButtonEvent\*)malloc(sizeof(ButtonEvent));  button->x\_min=x;  button->y\_min=y;  button->x\_max=x+w;  button->y\_max=y+h;  button->p=fp;  button->enable = 1;  //绘制按钮信息  fb\_draw\_rect(x,y,w,h,COLOR\_BACKGROUND);  fb\_draw\_border(x,y,w,h,color);  //绘制字符串到正中心  //获得字符串长度  int length=0;  while(str[length]!='\0') length++;  printf("length:%d",length);  int text\_x = x + (w - length \* fontSize/3)/2;  int text\_y = y + (h + fontSize)/2;  //绘制字体  fb\_draw\_text(text\_x,text\_y,str,fontSize,color);  //添加到按钮队列中  addButtonToList(button);  fb\_update();  return 0;  }  /\*按钮触发检测，调用触发的函数\*/  void buttonListClicked(int x,int y){    int i = 0;  printf("buttonList:\n");  for(i = 0;i < buttonList.tail;i++){  printf("button%d:x\_min:%d,x\_max:%d,y\_min:%d,y\_max:%d,enable:%d\n",i,buttonList.buttons[i]->x\_min,buttonList.buttons[i]->x\_max,buttonList.buttons[i]->y\_min,buttonList.buttons[i]->y\_max,buttonList.buttons[i]->enable);  }  printf("button check\n");  for(i = buttonList.tail-1;i>=0;i--){  printf("i=%d\n",i);  if(buttonList.buttons[i]->enable==0) continue;  if(x<buttonList.buttons[i]->x\_max&&x>buttonList.buttons[i]->x\_min&&y<buttonList.buttons[i]->y\_max&&y>buttonList.buttons[i]->y\_min){  /\*操作顺序：添加按钮--检测按钮--根据返回值调用按钮响应\*/  if(buttonList.buttons[i]->p==NULL) continue;  (\*(buttonList.buttons[i]->p))(buttonList.buttons[i]);  break;  }  }  printf("button check end\n");  }  /\*五子棋相关函数封装\*/  /\*总游戏初始化\*/  void initGame(){  //初始化棋盘  int i,j;  for(i = 0;i < 16;i ++){  for(j = 0;j < 16;j ++) board[i][j]=0;  }  //初始化状态  game\_state = state\_0;  //清空按钮队列  clearButtonList();  cauculateCircleFunc();  }  /\*初始化显示窗口\*/  void initScreen(){  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,COLOR\_BACKGROUND);  fb\_update();  }  /\*初始化棋盘绘制\*/  void drawChessBoard(){  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,COLOR\_BACKGROUND);  initChessboard();  fb\_update();  }  /\*棋盘绘制\*/  void initChessboard(){  int x = 242,y=30;  int i;  printf("begin--\n");  for(i = 0;i < 16;i++){  printf("i=%d\t",i);  fb\_draw\_line(x,y+36\*i,x+540,y+36\*i,COLOR\_LINE);  fb\_draw\_line(x+36\*i,y,x+36\*i,y+540,COLOR\_LINE);  }  //fb\_draw\_border(241,29,542,542,ORANGE);  }  /\*绘制节点函数值计算\*/  void cauculateCircleFunc(){  int i=0;  for(;i < 18;i++){  circleFunc[i]=(int)(sqrt(17\*17-i\*i)+0.5);  }  return;  }  /\*落子绘制，x,y为棋盘坐标\*/  void drawPiece(int x,int y,int color){  if(x<0||x>15||y<0||y>15) return;  board[x][y] = color;    x = 242+36\*x;  y = 30+36\*y;  int i=0;  for(;i<18;i++){  fb\_draw\_line(x+i,y-circleFunc[i],x+i,y+circleFunc[i],color);  fb\_draw\_line(x-i,y-circleFunc[i],x-i,y+circleFunc[i],color);  }  fb\_update();  }  /\*棋盘检测\*/  int checkBoard(int x,int y){  int curPieceColor = board[x][y];  int numOfPieces=1;  int i=x,j=y;  //left to right  while(i<15&&board[++i][j]==curPieceColor) numOfPieces++;  //right to left  i=x;  while(i>0&&board[--i][j]==curPieceColor) numOfPieces++;  if(numOfPieces == 5) return curPieceColor;    //up to down  i=x;numOfPieces=1;  while(j<15&&board[i][++j]==curPieceColor) numOfPieces++;  j=y;  while(j>0&&board[i][--j]==curPieceColor) numOfPieces++;  if(numOfPieces == 5) return curPieceColor;    //left-up to right-down  j=y;numOfPieces=1;  while(j>0&&i>0&&board[--i][--j]==curPieceColor) numOfPieces++;  i=x;j=y;  while(j<15&&i<15&&board[++i][++j]==curPieceColor) numOfPieces++;  if(numOfPieces == 5) return curPieceColor;    //left-down to right-up  i=x;j=y;numOfPieces=1;  while(j>0&&i<15&&board[++i][--j]==curPieceColor) numOfPieces++;  i=x;j=y;  while(j<15&&i>0&&board[--i][++j]==curPieceColor) numOfPieces++;  if(numOfPieces == 5) return curPieceColor;  return 0;  }  /\*触摸板事件控制模块，根据触摸位置和状态信息作出反应\*/  void touch\_event\_cb(int fd){  int type,x,y,finger;  int no\_x,no\_y;  type = touch\_read(fd, &x,&y,&finger);  switch(type){  case TOUCH\_PRESS:  //按钮检测  printf("%d,%d\n",x,y);  printf("game\_state=%d\n",game\_state);  if(game\_state == state\_6){  printf("game\_color=%d\n",game\_color);  /\*获得坐标位置信息，发送信息\*/  no\_x = (x-(242-18))/36;  no\_y = (y-(30-18))/36;  if(no\_x<0||no\_x>15||no\_y<0||no\_y>15) break;  if(board[no\_x][no\_y]!=0) break;  /\*绘制\*/  drawPiece(no\_x,no\_y,draw\_color);  fb\_update();  if(checkBoard(no\_x,no\_y)!=0){  /\*发送信息，状态修改\*/  bluetooth\_send(5,no\_x,no\_y,bluetooth\_fd);  game\_state = state\_8;  //添加按钮，本机获胜  //打印本机失败并创建重新开始的按钮  initButton(312,200,400,200,GREEN,"获胜",30,failureButtonEvent);  fb\_update();  }else{  /\*状态修改\*/  game\_state = state\_7;    /\*发送坐标信息\*/  bluetooth\_send(4,no\_x,no\_y,bluetooth\_fd);  }  }else if(game\_state == state\_7){}  else{  printf("checked\n");  buttonListClicked(x,y);  }  break;  case TOUCH\_ERROR:  printf("close touch fd\n");  break;  default:  return;  }  }  /\*五种信息类型：  1 : 准备信息 1  2 : 选择白棋 2  3 : 选择黑棋 3  4 : 落子坐标 4  5 : 胜利信息 5  \*/  /\*蓝牙信息接收模块\*/  void bluetooth\_tty\_event\_cb(int fd)  {  Message message;  int n;  n = myRead\_nonblock(fd, &message, sizeof(Message));  if(n <= 0) {  printf("close bluetooth tty fd\n");  //task\_delete\_file(fd);  //close(fd);  exit(0);  return;  }  //接收信息  printf("get message:type:%d\n",message.type);  switch(message.type){    case 1:  switch(game\_state){  case state\_0: /\*状态0 接收信息\*/  /\*对手完成准备，本机状态改变\*/  printf("state\_0 to state\_02\n");  game\_state = state\_02;  break;  case state\_01:  /\*完成准备，绘制选择界面\*/  printf("state\_01 to state\_1\n");  game\_state = state\_1;  drawSelect();  break;  default:  error("游戏状态转换，发送消息接收信息type1");  }  break;  case 2:  switch(game\_state){  case state\_1:  printf("state\_1 to state\_5\n");  game\_state = state\_5;  /\*白棋被选择\*/  /\*设置被选择信息，并将白棋选择按钮使能关闭\*/  buttonList.buttons[1]->enable=0;  initButton(512,200,200,200,WHITE,"已被选择",30,NULL);  fb\_update();  break;  case state\_2:  /\*清空按钮队列，绘制棋盘，转入状态6\*/  printf("state\_2 to state\_6\n");  game\_state = state\_6;  clearButtonList();  drawChessBoard();  break;  default:  error("游戏状态转换，接收消息type2");  }  break;  case 3:  switch(game\_state){  case state\_1:  printf("state\_1 to state\_4\n");  game\_state = state\_4;  /\*黑棋被选择\*/  /\*设置被选择信息，并将黑棋选择按钮使能关闭\*/  buttonList.buttons[0]->enable=0;  initButton(312,200,200,200,BLACK,"已被选择",30,NULL);  fb\_update();  break;    case state\_3:  /\*清空按钮队列，绘制棋盘，转入状态7\*/  printf("state\_3 to state\_7\n");  game\_state = state\_7;  clearButtonList();  drawChessBoard();  break;    default:  error("游戏状态转换，接收消息type3");  }  break;  case 4:  switch(game\_state){  case state\_7:  printf("state\_7 to state\_6\n");  game\_state = state\_6;  /\*分析相关信息\*/  /\*绘制棋子\*/  drawPiece(message.x,message.y,dis\_draw\_color); //队首棋颜色绘制    break;  default:  error("游戏状态转换，接收消息type4");  }  break;  case 5:  switch(game\_state){  case state\_7:  printf("state\_7 to state\_9\n");  game\_state = state\_9;  //打印本机失败并创建重新开始的按钮  initButton(312,200,400,200,RED,"失败",30,failureButtonEvent);  fb\_update();  break;  default:  error("游戏状态转换，接收消息type5");  }  break;  default:  error("游戏状态转换，消息类型不正确");  }  return;  }  /\*蓝牙信息发送\*/  void bluetooth\_send(int ctype,int cx,int cy,int fd){  //char buf[sizeof(Message)];  Message \*message = (Message\*)malloc(sizeof(Message));  message->type=ctype;  message->x=cx;  message->y=cy;  myWrite\_nonblock(fd, message, sizeof(Message));  printf("send message:type:%d,size:%d\n",message->type,sizeof(Message));  free(message);  }  int failureButtonEvent(ButtonEvent\* button){  //清空按钮队列  clearButtonList();  initScreen();  //绘制开始信息界面  //开始游戏，绘制开始界面，状态0  initGame();  //绘制准备界面,添加准备按钮  initButton(312,200,400,200,GREEN,"开始游戏",30,startButtonHandler);  game\_state = state\_0;  return 0;  }  /\*开始按钮事件处理\*/  int startButtonHandler(ButtonEvent\* button){  printf("startButtonHandler");  /\*游戏状态:state\_0-> state\_01 or state\_02 -> state\_1\*/  /\*修改游戏状态\*/  if(game\_state==state\_0){  printf("state\_0 to state\_01");  game\_state=state\_01;  clearButtonList();  //发送准备完成信息 1  bluetooth\_send(1,0,0,bluetooth\_fd);  /\*清空按钮队列，显示准备完成字样\*/  initScreen();  fb\_draw\_text(button->x\_min,button->y\_min,"等待另一玩家完成准备",20,COLOR\_TEXT);  fb\_update();  }  else if(game\_state==state\_02){  printf("state\_02 to state\_1");  game\_state=state\_1;  clearButtonList();  //绘制棋子选择界面，并添加两个按钮  drawSelect();  //send message  bluetooth\_send(1,0,0,bluetooth\_fd);  }  else{error("什么鬼啊，开始就出错？");}    return 0;  }  void drawSelect(){  initScreen();  //绘制棋子选择界面，并添加两个按钮  initButton(312,200,200,200,BLACK,"黑色",30,selectBlack);  initButton(512,200,200,200,WHITE,"白色",30,selectWhite);  fb\_update();  }  //黑棋选择按钮响应  int selectBlack(ButtonEvent\* button){  /\*颜色选中\*/  game\_color = 1;  draw\_color = BLACK;  dis\_draw\_color = WHITE;  //选择黑棋通讯 3  bluetooth\_send(3,0,0,bluetooth\_fd);  //状态修改  //state1 -> state2  if(game\_state == state\_1){  game\_state = state\_2;  //黑棋按钮修改为"已选择"，清空按钮队列  initButton(312,200,200,200,BLACK,"已选择",30,NULL);  fb\_update();  clearButtonList();  }  //state5 -> state6  else if(game\_state == state\_5){  game\_state = state\_6;  //绘制棋盘  initScreen();  drawChessBoard();  }  return 0;  }  //白棋选择按钮响应  int selectWhite(ButtonEvent\* button){  /\*颜色选中\*/  game\_color = 2;  draw\_color = WHITE;  dis\_draw\_color = BLACK;  //选择白棋通讯  bluetooth\_send(2,0,0,bluetooth\_fd);  //状态修改  if(game\_state==state\_1){  //白棋按钮修改为已选择，清空队列按钮  game\_state = state\_3;  //黑棋按钮修改为"已选择"，清空按钮队列  initButton(512,200,200,200,BLACK,"已选择",30,NULL);  fb\_update();  clearButtonList();  }else if(game\_state==state\_4){  game\_state = state\_7;  //绘制棋盘  initScreen();  drawChessBoard();  }  return 0;  }  void error(char str[]){  printf("error: %s",str);  exit(-1);  } |

# 参考文献

1. MIPS Technologies, Incorporated.MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set.2002年.
2. 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018年.
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京:清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京:清华大学出版社，2011年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |