**重庆大学Project报告**

Project题目： 共享白板系统

学 院： 计算机学院

专业班级： 计算机科学技术、计算机科学技术、

信息安全、信息安全

年 级： 2018

姓 名：陈江渝、许康乐、邹艳丽、周泽峻

学 号：20180007、20184969、20181759、20184228

完成时间： 2020 年 12 月 29 日

成 绩：

指导教师：

重庆大学教务处制

Project教师评定成绩表

A. 课程设计的成绩构成

|  |  |  |  |
| --- | --- | --- | --- |
| 课程设计项目 | 该项目标准分值 | 评定分值（100分） | |
| 项目完成质量（含检查） | 项目报告 |
| **项目1** | 10 | 80% | 20% |

B. 课程设计项目的成绩评定标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | 得分情况 | | | | |
| 优秀  (90～100) | 良好  (80～89) | 中等  (70～79 | 及格  60～69 | 不及格  0～59 |
| 参考标准 | 参考标准 | 参考标准 | 参考标准 | 参考标准 |
| 项目完成质量 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求，且有拓展  3.界面友好，Bug极少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求  3.界面良好，Bug较少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能较完善，基本满足课程设计要求  3.界面一般，Bug较多，无异常情况有处理  4.汇报检查时，讲解较清晰，完成演示，基本能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能基本完成，基本满足课程设计要求  3.界面一般，Bug很多，无异常情况有处理  4.汇报检查时，能基本讲清楚，主要功能能演示，基本能正确回答问题 | 1.抄袭或被抄袭以0分计，没有参与课程设计检查以0分计；  2.未能独立完成课程设计项目；  3.不能通过检查，不能进行课程设计汇报，主要功能无法演示，不能正确回答问题。 |
| 项目报告 | 1.结构严谨，逻辑清晰  2.文档规范，文字、图表表达清楚，用词专业，完全符合规范化要求  3.计算机打印  4.有完整的项目设计方案，采用的理论技术合理正确，结果及分析正确，功能有拓展  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑；  2. 文档规范，文字、图表表达较清楚，用词专业，符合规范化要求  3.计算机打印  4.有较完整的项目设计方案，采用的理论与技术正确，结果及分析正确  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑  2. 文档较规范，文字、图表表达较清楚，基本达到规范化要求  3.计算机打印  4.项目设计方案基本完整，采用的理论与技术基本正确，结果及分析基本正确。  5.报告内容与课程设计检查结果相符 | 1.结构基本合理,逻辑基本清楚  2. 文档较规范，文字、图表表达较清楚，勉强达到规范要求  3.计算机打印  4.项目设计方案有欠缺，采用的理论与技术基本可行，没有测试结果与结果分析  5.报告内容与课程设计检查结果相符 | 1.抄袭与被抄袭以0分计；  2.结构混乱，内容空泛，文字表达不清，错别字较多，达不到规范化要求  3.人工书写或计算机打印  4.报告内容与课程设计检查结果不相符 |

说明：评定分值要折算成项目标准分值。课程设计项目得分 = 课程设计项目的评定分值\*10%

指导教师评定成绩：

指导教师签名： 年 月 日

课题任务分工说明

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 角色 | 承担的任务 | 备注 |
| 1 |  |  | 组长 | （1）  （2）  （3）汇总各成员代码，并进行相应调整和部分改动 |  |
| 2 |  |  | 组员 | （1）  （2）  （3）  …… |  |
| 3 |  |  | 组员 | （1）  （2）  （3）  …… |  |
| 4 |  |  | 组员 | （1）  （2）  （3）  …… |  |

说明：应该把每个人承担的任务写清楚、仔细

目录

[1 需求分析 5](#_Toc60107234)

[2 系统设计 5](#_Toc60107235)

[2.1 系统分析 5](#_Toc60107236)

[2.2 系统设计 6](#_Toc60107237)

[2.2.1 应用层协议设计 6](#_Toc60107238)

[2.2.2 服务端行为说明 7](#_Toc60107239)

[2.2.3 客户端行为说明 8](#_Toc60107240)

[3 关键代码（技术）描述 9](#_Toc60107241)

[3.1 控制信息 9](#_Toc60107242)

[3.1.1 CType类 9](#_Toc60107243)

[3.1.2 CRequest类的主要代码 9](#_Toc60107244)

[3.1.3 CResponse类的主要代码 9](#_Toc60107245)

[3.2 画图信息 9](#_Toc60107246)

[3.2.1 PType类 9](#_Toc60107247)

[3.2.2 SType类 9](#_Toc60107248)

[3.2.3 PData类 9](#_Toc60107249)

[3.2.4 PDataBody各个类 9](#_Toc60107250)

[3.3 服务端 9](#_Toc60107251)

[3.3.2公共变量和函数 9](#_Toc60107252)

[3.3.2 Server类 9](#_Toc60107253)

[3.3.3 User类 9](#_Toc60107254)

[3.4 客户端 9](#_Toc60107255)

[3.4.1 ClientConn类 9](#_Toc60107256)

[3.4.2 Client类 9](#_Toc60107257)

[3.4.3 WhiteBoardGUI文件中的提交画图信息函数 9](#_Toc60107258)

[4 系统测试运行 10](#_Toc60107259)

[4.1 服务端 10](#_Toc60107260)

[4.2 客户端 10](#_Toc60107261)

[4.2.1 开启两个客户端，分别连接服务器 10](#_Toc60107262)

[4.2.2 进入主界面 11](#_Toc60107263)

[4.2.2 打开白板 11](#_Toc60107264)

[5 总结 11](#_Toc60107265)

# 1 需求分析

**共享白板系统**

* 实现基于服务器转发的多点间的文字、作图等信息的实时同步共享。
* 作图包括笔刷、划线、矩形、圆形等，可以调节粗细、颜色、清屏。
* 实时同步在线成员列表。
* 所有参与者均可发起共享或取消已有的共享。
* 其他参与者要能及时展示出发起或取消的共享信息。
* 所有参与者实时同步白板内容。
* 新上线的用户可以显示历史白板记录。
* 基于SOCKET编程实现
* 传输层可采用TCP或UDP协议
* 能够具有良好的图形界面。

# 2 系统设计

## 2.1 系统分析

系统实现的是一个基于服务器转发的多点间的文字、作图等信息的实时同步共享的共享白板系统。服务端和客户端基于SOCKET编程实现，传输层采用TCP协议。

服务端通过多线程处理多个客户端的连接，主要负责解析客户端的请求，并进行相应的处理。请求/响应主要有四种类型：

1. **分配用户ID**

新用户连接时，向其发放ID作为该客户端的唯一标识。

1. **在线用户信息**

每当有用户连接或断开时，向所有在线用户发送在线用户信息，实现在线成员列表的同步。

1. **画图数据**

有用户进行了绘制相关操作时，发送给服务端，由服务端转发给其余在线用户。

1. **用户断开连接**

有用户退出时，通知服务端，服务端更新在线用户的记录，并通知所有在线用户。

## 2.2 系统设计

### 2.2.1 应用层协议设计

由于高级语言的特点，数据编码传输时，我们将字段转换为字符串，再对其进行utf-8编码得到字节流。

#### 控制信息 CRequest, CResponse

控制信息分为请求和回复，客户端给服务端发送CRequest请求，服务端返回给客户端CResponse回复，二者结构相同，如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
|  | Header | | Body |
| 长度 | 1 byte | 3 byte | 可变 |
| 字段名 | cType | bodyLen |  |
| 说明 | 控制信息类型，  CRequest时，cType可取值为：  2 - 画图数据PData  3 - 断开连接  CResponse时，cType可取值为：  0 - 分配用户ID  1 - 在线用户信息  2 - 画图数据Pdata | 数据部分的长度，三位十进制数。 | 数据 |

编码规则：

头的各个字段直接转换为对应字符串，再拼接body字符串，一并转换为字节流。

#### 绘图信息 pData

绘图信息装入控制信息的Body中进行传输。

结构如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Header | | | Body |
| 长度 | 1 byte | 7 byte | 7 byte | 可变 |
| 字段名 | pType | foreColor | backColor |  |
| 说明 | 绘制类型，  0 - 笔刷  1 - 形状  2 - 文字  3 - 橡皮  4 - 清屏 | 前景色，  十六进制颜色字符串，  如'#000000' | 背景色，  十六进制颜色字符串，  如'#000000' | 数据 |

编码规则：

头的各个字段转换为字符串，再加上body部分的字符串，一起用符号’\_’连接在一起。

Body部分的结构：

* 笔刷PDataBrush：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | st | ed | width |
| 说明 | 起点坐标 | 终点坐标 | 笔刷粗细 |

* 形状PDataShape：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | sType | st | ed | width |
| 说明 | 形状类型，  0 - 直线  1 - 矩形  2 - 圆形 | 起点坐标 | 终点坐标 | 笔刷粗细 |

* 文本PDataText：

|  |  |  |
| --- | --- | --- |
| 字段名 | content | pos |
| 说明 | 文本内容 | 坐标 |

* 橡皮PDataEraser：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | st | ed | width |
| 说明 | 起点坐标 | 终点坐标 | 笔刷粗细 |

编码规则：

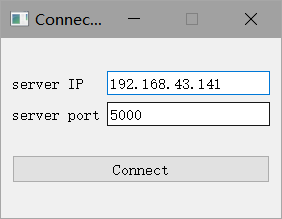
坐标字段（st, ed, pos）的两个分坐标分别转换为字符串，其他字段直接转换为字符串，所有字符串一起用符号’^’连接在一起。

### 2.2.2 服务端行为说明

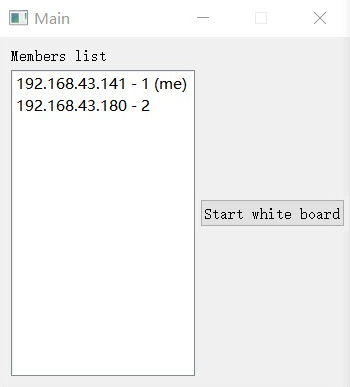
1. 服务端需要维护在线用户信息表、下一个可用ID和历史画图记录。
2. 在设置好的ip、端口上建立使用TCP连接的socket。
3. 每收到一个新的连接，就开启一个客户端处理线程（User类）处理该客户端的事务。
4. 新用户连接时，为其分配下一个可用ID作为该客户端的唯一标识，加入在线用户信息表中；同时向所有用户发送在线用户信息；此外，发送历史画图记录给新连接的用户。
5. 有用户发送含有画图信息的CRequest时，服务端解包出其中的数据部分，记入历史画图记录，然后封装入CReponse，转发给其他所有在线用户。
6. 收到用户发来的断线信息，或检测到连接不可用时，将该用户从在线用户信息表中删除，并向所有用户发送新的在线用户信息。

### 2.2.3 客户端行为说明

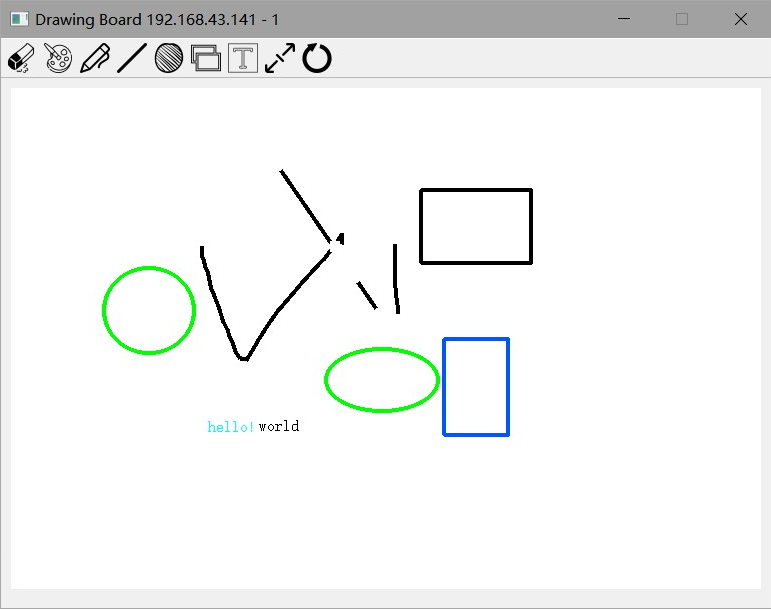
1. 连接服务器界面，按照GUI提示输入服务器ip和端口号，尝试与服务器建立连接：



1. 连接成功后，获取到服务器发来的ID，获取到在线用户信息，历史画图记录。
2. 显示主窗口页面，其中包含自己的ID和在线成员列表：



1. 用户打开白板后，先显示历史画图记录（如果有）
2. 用户进行绘图操作时，不仅要在本地白板上绘制，同时还要生成相应的画图信息PData，发送给服务端。
3. 获取到服务端发送的画图信息时，在白板上绘制出：



1. 用户关闭客户端时，发送断开信号给服务器。

# 3 关键代码（技术）描述

## 3.1 控制信息

### 3.1.1 CType类

**class** CType(Enum):  
 ID = 0 *# 用户的ID* USERINFOS = 1 *# 在线用户信息* PDATA = 2 *# 画图数据* DISCONNECT = 3 *# 断开连接* NOOP = 4 *# 不可用* **def** \_\_str\_\_(self):  
 *"""The value of the Enum member."""* **return** str(self.value)  
  
 **def** \_\_bytes\_\_(self):  
 **return** str(self.value).encode(ENCODING)

### 3.1.2 CRequest类的主要代码

**class** CRequest:  
 HEADER\_LEN = 4  
 **def** \_\_init\_\_(self, ctype=**None**, bodyLen=**None**, body=**None**):  
 *# header* self.ctype = ctype *# 1 byte* self.bodyLen = bodyLen *# 3 byte  
 # body* self.body = body  
   
 @staticmethod  
 **def** mkRequest(ctype, body=**None**):  
 **if** body:  
 bodyLen = **'%03d'** % len(str(body))  
 **else**:  
 bodyLen = **'000'  
 return** CRequest(ctype, bodyLen, body)  
  
 **def** \_\_str\_\_(self):  
 *# bodyLen 占 3 位* l = [str(self.ctype), str(self.bodyLen), str(self.body) **if** self.body **else ''**]  
 **return** CSEP.join(l)  
  
 **def** encode(self) -> bytes:  
 **return** str(self).encode(ENCODING)  
  
 @staticmethod  
 **def** decodeHeader(headers: bytes):  
 l = headers.decode(ENCODING)  
 type = CType(int(l[0]))  
 bodyLen = int(l[1:CResponse.HEADER\_LEN])  
 **return** CRequest(type, bodyLen=bodyLen)  
  
 **def** decodeBody(self, bodyData):  
 self.body = bodyData  
 **return** self

### 3.1.3 CResponse类的主要代码

**class** CResponse:  
 HEADER\_LEN = 4  
 **def** \_\_init\_\_(self, ctype=**None**, bodyLen=**None**, body:str=**None**):  
 *# header* self.ctype = ctype *# 1 byte* self.bodyLen = bodyLen *# 3 byte  
 # body* self.body = body

@staticmethod  
 **def** mkResponse(ctype: CType, body: str):  
 **if** body:  
 bodyLen = **'%03d'** % len(str(body))  
 **else**:  
 bodyLen = **'000'  
 return** CResponse(ctype, bodyLen, body)  
  
 @staticmethod  
 **def** id(id: int):  
 *# 每当新用户连接时，发送给该用户* **return** CResponse.mkResponse(CType.ID, str(id))  
  
 @staticmethod  
 **def** userInfos(usersdict: dict):  
 *# 每当新用户连接时或有用户断开时，发送给所有用户* **return** CResponse.mkResponse(CType.USERINFOS, json.dumps(usersdict))  
  
 @staticmethod  
 **def** pData(pDataBytes: bytes):  
 **return** CResponse.mkResponse(CType.PDATA, pDataBytes.decode(ENCODING))  
  
 **def** \_\_str\_\_(self):  
 l = [str(self.ctype), str(self.bodyLen), str(self.body)]  
 **return** CSEP.join(l)  
  
 **def** encode(self) -> bytes:  
 **return** str(self).encode(ENCODING)  
  
 @staticmethod  
 **def** decodeHeader(headers: bytes):  
 l = headers.decode(ENCODING)  
 type = CType(int(l[0]))  
 bodyLen = int(l[1:CResponse.HEADER\_LEN])  
 **return** CResponse(type, bodyLen=bodyLen)  
  
 **def** decodeBody(self, bodyDataBytes: bytes):  
 bodyData = bodyDataBytes.decode(ENCODING)  
 **if** self.ctype == CType.PDATA:  
 self.body = PData.decodeFromStr(bodyData)  
 **elif** self.ctype == CType.ID:  
 self.body = bodyData  
 **elif** self.ctype == CType.USERINFOS:  
 self.body = json.loads(bodyData)  
 **return** self

## 3.2 画图信息

### 3.2.1 PType类

**class** PType(Enum):  
 *"""绘制类型，0-笔刷，1-形状，2-文字，3-橡皮, 4-清屏, 5-不可用"""* BRUSH = 0  
 SHAPE = 1  
 TEXT = 2  
 ERASER = 3  
 CLS = 4  
 NA = 5  
 **def** \_\_str\_\_(self):  
 **return** str(self.value)

### 3.2.2 SType类

**class** SType(Enum):  
 *"""形状类型，0-直线，1-矩形，2-圆，3-不适用（不是画形状）"""* LINE = 0  
 RECT = 1  
 CIRCLE = 2  
 NA = 3  
 **def** \_\_str\_\_(self):  
 **return** str(self.value)

### 3.2.3 PData类

**class** PData:  
 *"""传输的数据结构"""* SEP = **'\_'  
 def** \_\_init\_\_(self, pType: PType, foreColor: QColor, backColor: QColor, body:Type[TPDataBody]=**None**):  
 *"""* **:param** *pType: 绘制类型，0-刷子，1-形状，2-文字，3-橡皮, 4-清屏* **:param** *foreColor: 前景色，编码后为十六进制颜色信息，如'#000000'* **:param** *backColor: 背景色，编码后为十六进制颜色信息，如'#000000'  
 """  
 # Header* self.pType = pType  
 self.foreColor = foreColor  
 self.backColor = backColor  
  
 *# Body* self.body = body

**def** \_\_str\_\_(self):  
 *# 转为字符串* body = str(self.body) **if** self.body **else ''** l = [str(self.pType), str(self.foreColor.name()), str(self.backColor.name()), body]  
 **return** PData.SEP.join(l)  
  
 @staticmethod  
 **def** decodeFromBytes(pdataBytes: bytes)->TPData:  
 **return** PData.decodeFromStr(pdataBytes.decode(ENCODING))  
  
 @staticmethod  
 **def** decodeFromStr(pdata: str)-> TPData:  
 l = pdata.split(PData.SEP)  
 pType = PType(int(l[0]))  
 foreColor = QColor(l[1])  
 backColor = QColor(l[2])  
 body = **None  
 if** pType == PType.BRUSH:  
 body = PDataBrush.decodeFromStr(l[3])  
 **elif** pType == PType.SHAPE:  
 body = PDataShape.decodeFromStr(l[3])  
 **elif** pType == PType.TEXT:  
 body = PDataText.decodeFromStr(l[3])  
 **elif** pType == PType.ERASER:  
 body = PDataEraser.decodeFromStr(l[3])  
   
 **return** PData(pType, foreColor, backColor, body)

### 3.2.4 PDataBody各个类

#### 1. 基类PDataBody:

**class** PDataBody:  
 *"""传输的数据结构的body部分"""* SEP = **'^'**

#### 2. 刷子PDataBrush：

**class** PDataBrush(PDataBody):  
 *"""刷子类型的数据结构"""* **def** \_\_init\_\_(self, st: Tuple, ed: Tuple, width):  
 *"""* **:param** *st: 起点坐标* **:param** *ed: 终点坐标* **:param** *width: 笔刷粗细  
 """* self.st = st  
 self.ed = ed  
 self.width = width  
  
 **def** \_\_str\_\_(self):  
 l = [str(self.st[0]), str(self.st[1]), str(self.ed[0]), str(self.ed[1]), str(self.width)]  
 **return** PDataBody.SEP.join(l)  
@staticmethod  
 **def** decodeFromStr(body: str) -> TPDataBody:  
 l = body.split(PDataBody.SEP)  
 st = (int(l[0]), int(l[1]))  
 ed = (int(l[2]), int(l[3]))  
 width = int(l[4])  
 **return** PDataBrush(st, ed, width)

#### 3. 形状PDataShape：

**class** PDataShape(PDataBody):  
 *"""形状类型的数据结构"""* **def** \_\_init\_\_(self, sType: SType, st: Tuple=**None**, ed: Tuple=**None**, width=**None**):  
 *"""* **:param** *sType: 形状类型，0-直线，1-矩形，2-圆* **:param** *st: 起点* **:param** *ed: 终点* **:param** *width: 笔刷粗细  
 """* self.sType = sType  
 self.st = st  
 self.ed = ed  
 self.width = width  
   
 **def** \_\_str\_\_(self):  
 l = [str(self.sType), str(self.st[0]), str(self.st[1]), str(self.ed[0]), str(self.ed[1]), str(self.width)]  
 **return** PDataBody.SEP.join(l)  
@staticmethod  
 **def** decodeFromStr(body: str) -> TPDataBody:  
 l = body.split(PDataBody.SEP)  
 sType = SType(int(l[0]))  
 st = (int(l[1]), int(l[2]))  
 ed = (int(l[3]), int(l[4]))  
 width = int(l[5])  
 **return** PDataShape(sType, st, ed, width)

#### 4. 文本PDataText：

**class** PDataText(PDataBody):  
 *"""文字类型的数据结构"""* **def** \_\_init\_\_(self, content: str, pos: Tuple):  
 *"""* **:param** *content: 文本内容* **:param** *pos: 位置  
 """* self.content = content  
 self.pos = pos  
   
 **def** \_\_str\_\_(self):  
 l = [self.content, str(self.pos[0]), str(self.pos[1])]  
 **return** PDataText.SEP.join(l)  
@staticmethod  
 **def** decodeFromStr(body: str) -> TPDataBody:  
 l = body.split(PDataBody.SEP)  
 content = l[0]  
 pos = (int(l[1]), int(l[2]))  
 **return** PDataText(content, pos)

#### 5. 橡皮PDataEraser：

**class** PDataEraser(PDataBody):  
 *"""橡皮类型的数据结构"""* **def** \_\_init\_\_(self, st: Tuple, ed: Tuple, width):  
 *"""* **:param** *st: 起点坐标* **:param** *ed: 终点坐标* **:param** *width: 橡皮笔刷粗细  
 """* self.st = st  
 self.ed = ed  
 self.width = width  
   
 **def** \_\_str\_\_(self):  
 l = [str(self.st[0]), str(self.st[1]), str(self.ed[0]), str(self.ed[1]), str(self.width)]  
 **return** PDataBody.SEP.join(l)  
@staticmethod  
 **def** decodeFromStr(body: str) -> TPDataBody:  
 l = body.split(PDataBody.SEP)  
 st = (int(l[0]), int(l[1]))  
 ed = (int(l[2]), int(l[3]))  
 width = int(l[4])  
 **return** PDataEraser(st, ed, width)

## 3.3 服务端

### 3.3.2公共变量和函数

*# 存客户端线程实例*users = {} *# type: Dict[str, User] # Dict[id, User]*userInfos = {} *# Dict[id, ip]  
# 存服务器的数据，用于图像的复现*logs = [] *# type: List[CResponse]*

**def** validIp(ip: str) -> bool:  
 *"""检查IP地址是否合法"""* pattern = **r'^((2(5[0-5]|[0-4]\d))|[0-1]?\d{1,2})(\.((2(5[0-5]|[0-4]\d))|[0-1]?\d{1,2})){3}$'  
 return** re.match(pattern, ip) **is not None  
  
def** validPort(port: str) -> bool:  
 *"""检查port是否合法"""  
 # 0~65535* **return** port.isnumeric() **and** 0<=int(port)<=65535

### 3.3.2 Server类

**class** Server(socket.socket):  
 *"""连接客户端的类，封装socket"""* nextUserId = 1 *# 为用户分配id* lock = Lock()  
  
 **def** \_\_init\_\_(self, host=**'127.0.0.1'**, port=5000):  
 self.host = host  
 self.port = port  
 **if not** validIp(self.host) **or not** validPort(str(self.port)):  
 print(**f"invalid server address {self.host}:{self.port}"**)  
 **return** *# 新建一个socket* super().\_\_init\_\_() *# create ip socket* self.bind((self.host, self.port))  
 print(**f"hosting at {self.host}:{self.port}"**)  
 self.listen(5) *# 开启监听，设置允许等待的连接个数=5* self.serveForever()  
  
 *# 用多线程以支持多个连接* **def** serveForever(self):  
 **while True**:  
 conn, addr = self.accept() *# 阻塞，每收到一个连接就唤醒  
 # 新建一个处理该连接的ClientObj线程* user = User(conn, self.nextUserId, addr)  
 print(**'\nconnected by'**, addr, **'-'**, user.id)  
 *# 记入字典* users[user.id] = user  
 userInfos[user.id] = user.ip  
 *# 启动对应的线程，转发数据* user.start()  
 *# 每当新用户连接时，发送id给该用户* user.assignId()  
 *# 有新用户连接时，发送userinfos给所有用户* **for** id, usera **in** users.items():  
 usera.sendUserInfos()  
 *# 有新用户连接时，发送历史图像给该用户* user.sendLogs()  
  
 @classmethod  
 **def** mutexIncUserId(cls):  
 *# 互斥修改nextUserId* cls.lock.acquire()  
 cls.nextUserId += 1  
 cls.lock.release()  
  
 @classmethod  
 **def** mutexDecUserId(cls):  
 *# 互斥修改nextUserId* cls.lock.acquire()  
 cls.nextUserId -= 1  
 cls.lock.release()

### 3.3.3 User类

**class** User(Thread):  
 *# 为每个连接的客户端创建一个线程实例，用于并行处理所有客户端发来的消息* **def** \_\_init\_\_(self, conn: socket.socket, id, addr):  
 super().\_\_init\_\_()  
 self.conn = conn  
 self.id = id  
 self.alive = **True** self.ip = addr[0]  
  
 **def** assignId(self):  
 cResp = CResponse.id(self.id)  
 print(**"\nrespond to"**, self.ip, **'-'**, self.id, **":"**)  
 print(cResp.print())  
 self.sendCResp(cResp)  
 Server.mutexIncUserId()  
  
 **def** sendLogs(self):  
 print(**"\nsend logs to user"**, self.id, **":"**)  
 **for** cResp **in** logs:  
 self.sendCResp(cResp)  
  
 **def** forwardContent(self, bodyBytes: bytes):  
 *# 把内容封装到response里，转发给除了自己之外的所有用户* cResp = CResponse.pData(bodyBytes)  
 *# 记入log* logs.append(cResp)  
 **if** PData.decodeFromBytes(bodyBytes).isCls():  
 logs.clear()  
 print()  
 **for** id, user **in** users.items():  
 **if** id != self.id:  
 print(**"forward pdata to user "**, id)  
 user.sendCResp(cResp)  
 print(cResp.print())  
  
 **def** sendUserInfos(self):  
 cResp = CResponse.userInfos(userInfos)  
 print(**"\nsend to"**, self.ip, **'-'**, self.id, **":"**)  
 print(cResp.print())  
 self.sendCResp(cResp)  
  
 **def** sendCResp(self, cResp: CResponse):  
 cRespBytes = cResp.encode()  
 **try**:  
 print(**"send raw"**, cRespBytes)  
 self.conn.sendall(cRespBytes)  
 **except** ConnectionError:  
 print(**"connection error, closing"**)  
 self.handleDisconnRequest()  
  
 **def** recvCReq(self) -> CRequest:  
 *# 一次接收一个CReq*data = self.conn.recv(CRequest.HEADER\_LEN)  
 **if** data != **b''**:  
 cReq = CRequest.decodeHeader(data)  
 data = self.conn.recv(cReq.bodyLen)  
 cReq.decodeBody(data)  
 **return** cReq  
 **return** CRequest(CType.NOOP)  
  
 **def** handleCtrlRequest(self, cReq: CRequest):  
 **if** cReq.ctype == CType.PDATA:  
 self.forwardContent(cReq.body)  
 **elif** cReq.ctype == CType.DISCONNECT:  
 self.handleDisconnRequest()  
  
 **def** handleDisconnRequest(self):  
 *# 连接关闭* self.conn.close()  
 **del** users[self.id]  
 **del** userInfos[self.id]  
 print(**"connection from"**, self.ip, **'-'**, self.id, **"is closed"**)  
 Server.mutexDecUserId()  
 self.alive = **False** *# 有用户断开时，发送userinfos给所有用户* **for** id, user **in** users.items():  
 user.sendUserInfos()  
  
 **def** run(self):  
 **while** self.alive:  
 **try**:  
 print(**"receive from"**, self.ip, **'-'**, self.id)  
 cReq = self.recvCReq() *# 阻塞，收到数据后唤醒* print(**"receive"**, cReq.print())  
 self.handleCtrlRequest(cReq)  
 **except** ConnectionError:  
 print(**"connection error, closing"**)  
 self.handleDisconnRequest()

## 3.4 客户端

### 3.4.1 ClientConn类

**class** ClientConn(socket.socket):  
 *"""连接服务器的类，封装socket"""* \_poll\_all\_users\_interval = 500  
  
 **def** \_\_init\_\_(self):  
 socket.socket.\_\_init\_\_(self) *# create tcp/ip socket* self.isAlive = **False** self.serverIp = **None** self.serverPort = **None  
  
 def** initConn(self, ip, port):  
 *"""连接服务器，返回是否成功"""* **try**:  
 self.connect((ip, port))  
 **except** socket.error **as** e:  
 print(**"Error:"**, e)  
 **return False** self.isAlive = **True** self.serverIp = ip  
 self.serverPort = port  
 print(**f"connected to server at {self.serverIp}:{self.serverPort}"**)  
 **return True** @property  
 **def** hostIp(self):  
 *# 获取本机IP* **return** self.getsockname()[0]  
  
 **def** disconnect(self):  
 **try**:  
 self.sendall(CRequest.disconnect().encode())  
 **except** ConnectionError:  
 **pass** self.shutdown(socket.SHUT\_RDWR)  
 self.close()  
 self.isAlive = **False  
  
 def** sendCReq(self, cReq: CRequest):  
 *# 一次发送一个CReq* cDataBytes = cReq.encode()  
 **try**:  
 self.sendall(cDataBytes)  
 print(**"sent"**, cReq.print())  
 **except** ConnectionError:  
 print(**"send error"**)  
 print(**"connection to server is unavailable, closing program..."**)  
 self.shutdown(socket.SHUT\_RDWR)  
 self.close()  
 self.isAlive = **False  
  
  
 def** recvCResp(self) -> CResponse:  
 *# 一次接收一个CResp* data = self.recv(CResponse.HEADER\_LEN)  
 cResp = CResponse.decodeHeader(data)  
 data = self.recv(cResp.bodyLen)  
 cResp.decodeBody(data)  
 print(**"receive raw"**, cResp)  
 **return** cResp

### 3.4.2 Client类

#### 主要成员

self.conn = ClientConn()

self.connWind = ConnectWindow(self.conn) *# 使用lambda函数防止直接获取还未定义的main*

self.board = WhiteBoardWindow(self.conn, self.id, callback=**lambda**: self.main.toggleWhiteBoard()) *# 使用lambda函数防止直接获取还未定义的main*

self.main = Main(self.board, self.conn, self.id)

#### Client的线程执行函数

该函数用于接收服务器的信息并执行相应行为。

**def** run(self):  
 print(**"running thread"**)  
 **while** self.conn.isAlive:  
 **try**:  
 cResp = self.conn.recvCResp()  
 **except** OSError **as** e:  
 *# 关闭socket连接后阻塞中的recv会触发OSError* print(**"recv error"**)  
 print(**"connection to server is unavailable, closing program..."**)  
 **break** print(**"receive"**, cResp.print())  
 **if** cResp.ctype == CType.USERINFOS:  
 *# 收到用户信息* userinfodict = cResp.body  
 l = []  
 **for** id, ip **in** userinfodict.items():  
 **if** id == self.id:  
 l.append(**f"{ip} - {id} (me)"**)  
 **else**:  
 l.append(**f"{ip} - {id}"**)  
 self.main.setUserInfos(l)  
  
 **elif** cResp.ctype == CType.PDATA:  
 *# 收到PDATA* pData = cResp.body  
 self.board.wb.paintFromMsg(pData)  
  
 *# 执行到这里说明连接断开了，关闭窗口* self.main.forceClose()

### 3.4.3 WhiteBoardGUI文件中的提交画图信息函数

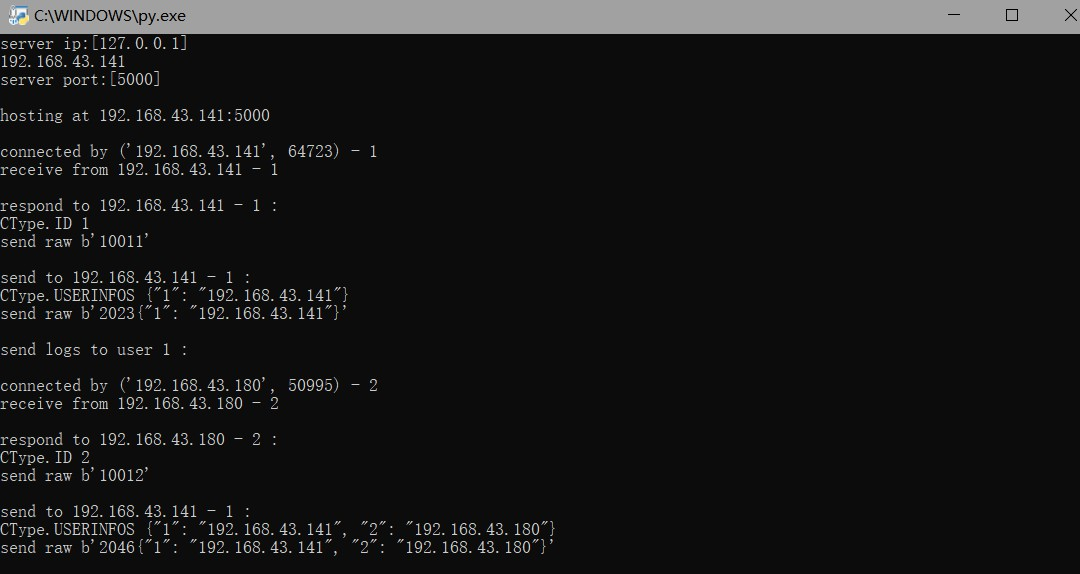
**def** sendCReq(self, pData=**None**):  
 **if** pData:  
 cReq = CRequest.pData(pData)  
 **else**:  
 cReq = CRequest.pData(self.pData)  
 self.conn.sendCReq(cReq)  
 **if not** self.conn.isAlive:  
 **return**

# 4 系统测试运行

使用

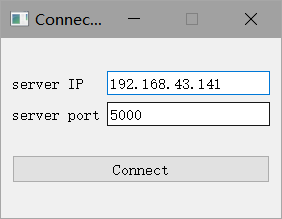
## 4.1 服务端

在192.168.43.141:5000成功启动，并按预期接收和响应数据。



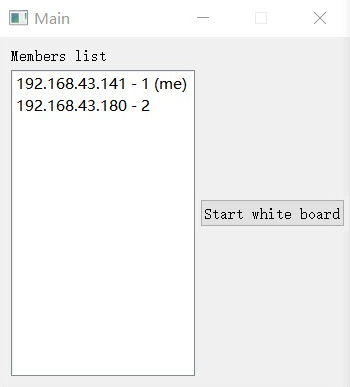
## 4.2 客户端

### 4.2.1 开启两个客户端，分别连接服务器



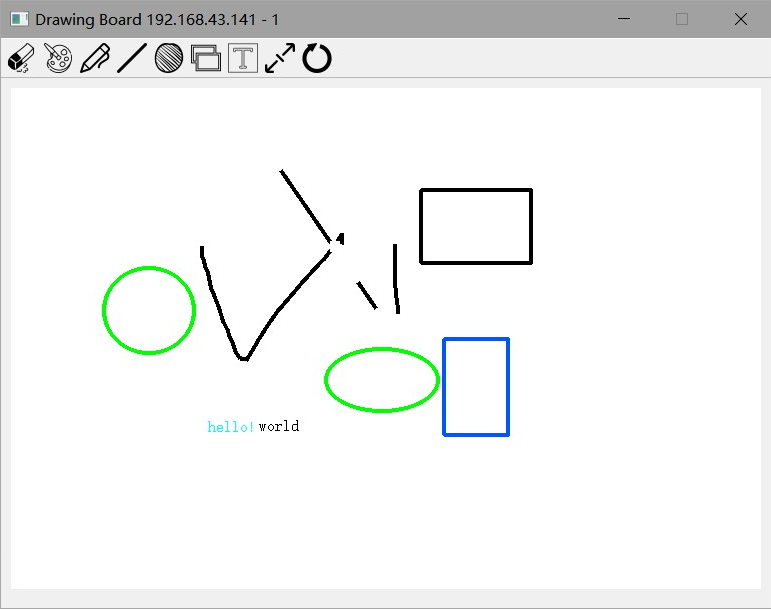
### 4.2.2 进入主界面

可以看到两个客户均在线



### 4.2.2 打开白板

可以看到绘制的内容成功共享到另一个客户端的白板上，Server端也有相应的转发记录



# 5 总结

* 学会了socket如何建立tcp连接；学会了如何通过socket进行数据的编码发送和接收解码；学会了socket相关的错误处理；学会了如何使用多线程处理多个连接。
* 掌握了如何设计协议，主要在于设计字段结构和代表的含义
* 学会了python的GUI设计，实现了白板