****

网络编程课程设计

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 信息安全 |
| **班 级** | 16273611 |
| **学 号** | 16271112 |
| **学生姓名** | 费朝烨 |
| **教师姓名** | 张祯 |
| **完成日期** | 2018/10/19 |
| **成 绩** |  |

**实验一 基于UDP的C/S程序设计**

1. **实验目的**
2. 熟悉UDP协议的原理和特点
3. 掌握基于UDP协议的应用层协议设计方法。
4. 掌握基于UDP协议的C/S网络应用程序设计和实现方法。
5. **实验内容**

**以下题目任选其一，或在征求指导老师同意的情况下，完成难度相似的基于UDP的网络应用。**

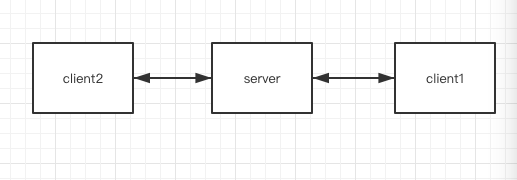
1. 编写基于UDP的聊天室程序，实现多人聊天功能。自己设计应用协议，要求实现以下功能：
2. 用户注册
3. 用户登录
4. 公聊
5. 私聊
6. 编写基于UDP的文件传输程序（可以参考TFTP协议，不能照抄），实现上传文件到服务器和从服务器下载文件，要求实现以下功能：
7. 获取文件列表
8. 上传文件
9. 下载文件

注意测试两个客户端同时进行上传和下载的情况

1. 基于UDP的屏幕广播程序，将服务器的桌面通过UDP的广播或多播，发送给对应的客户端程序，客户端接收并显示服务器的桌面信息（注意屏幕图片大小超过UDP的最大报文）。
2. **程序设计思路**
3. **所选题目说明：**

所选题目1，设计基于udp的聊天室程序，由于udp并不是基于连接的所以在聊天中都可以需要将其状态保存。

1. **网络应用拓扑结构**



每个client都与server同时连接，server端保存有所有的在连接的client的ip与端口，在client公播时，向所有client发送消息，当私聊是，server收到请求，并向所请求的client发送消息

1. **应用层协议设计**

**整个协议基于udp进行传输，在无连接的基础上**基于json设计请求与响应，主要包括login，register，send\_message，logout，recv\_message等请求以及相应。

login json

request

{

"method":"login",

"parm":{

"username":"xxxxx",

"password":"xxxxx",

}

}

response

{

"status":"success",

"token":"xxxxxx"

}

or

{

"status":"error"

}

register json

request

{

"method":"register",

"parm":{

"username":"xxxxx",

"password":"xxxxx"

}

}

response

{

"status":"success",

"token":"xxxxxxxx"

}

or

{

"status":"error"

}

send\_message

request

{

"method":"send"

"parm":{

"token":"xxxxx",

"message":"xxxxx",

"to\_user":"x"

}

}

response

{

"status":"send seccuss",

}

or

{

"status":"send error"

}

recive\_message

response

{

"from\_user":"",

"message":"xxxx",

}

请求以及回复都进行编码。

1. **所选用的Python库介绍**

socket: This module provides access to the BSD socket interface. It is available on all modern Unix systems, Windows, Mac OS X, BeOS, OS/2, and probably additional platforms.

pyqt5: Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development.

Json: [json](https://docs.python.org/2/library/json.html#module-json) exposes an API familiar to users of the standard library [marshal](https://docs.python.org/2/library/marshal.html#module-marshal) and [pickle](https://docs.python.org/2/library/pickle.html#module-pickle) modules.

Logging: This module defines functions and classes which implement a flexible event logging system for applications and libraries.

Multiprocessing: [**multiprocessing**](https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) is a package that supports spawning processes using an API similar to the [**threading**](https://docs.python.org/2/library/threading.html#module-threading) module. The [**multiprocessing**](https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) package offers both local and remote concurrency, effectively side-stepping the [Global Interpreter Lock](https://docs.python.org/2/glossary.html#term-global-interpreter-lock) by using subprocesses instead of threads. Due to this, the [**multiprocessing**](https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing) module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

sqlite3: The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by [**PEP 249**](https://www.python.org/dev/peps/pep-0249).

1. **程序源代码**

**注意源代码要有详细的注释。 同学们提交的每个程序都应该遵循Honor Code（诚实代码保证）的要求。**

**请大家特别注意一定要在每个程序首部的注释中加上以下保证：**

**# 我真诚地保证：**

**# 我自己独立地完成了整个程序从分析、设计到编码的所有工作。**

**# 如果在上述过程中，我遇到了什么困难而求教于人，那么，我将在程序实习报告中**

**# 详细地列举我所遇到的问题，以及别人给我的提示。**

**# 在此，我感谢 XXX, …, XXX对我的启发和帮助。下面的报告中，我还会具体地提到**

**# 他们在各个方法对我的帮助。**

**# 我的程序里中凡是引用到其他程序或文档之处，**

**# 例如教材、课堂笔记、网上的源代码以及其他参考书上的代码段,**

**# 我都已经在程序的注释里很清楚地注明了引用的出处。**

**# 我从未没抄袭过别人的程序，也没有盗用别人的程序，**

**# 不管是修改式的抄袭还是原封不动的抄袭。**

**# 我编写这个程序，从来没有想过要去破坏或妨碍其他计算机系统的正常运转。**

**# 费朝烨**

1. **服务器端源码**
2. #!/usr/bin/env python  
   # -\*- coding: utf-8 -\*-  
   *"""  
   \_\_title\_\_ = ''  
   \_\_author\_\_ = 'ralph'  
   \_\_mtime\_\_ = '2018/10/16'  
   """*import socket  
   import logging  
   import json  
     
   from multiprocessing import Process  
   from database import database  
     
   logging.basicConfig(level=logging.INFO)  
     
   MAX\_SIZE = 1024  
     
     
   class servers():  
    def \_\_init\_\_(self):  
    global MAX\_SIZE  
    ip = "127.0.0.1"  
    port = 23333  
      
    self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
    self.server\_socket.bind((ip, port))  
      
    self.db = database()  
      
    logging.info("socket create success")  
    logging.info("bind hostname is (" + ip + "," + str(port) + ")")  
      
    self.server\_connects = []  
    self.server\_conn\_hostname = []  
      
    self.server\_listen()  
      
    def server\_listen(self):  
    while True:  
    try:  
    logging.info("program is listening...")  
    recv\_data, addr = self.server\_socket.recvfrom(MAX\_SIZE)  
    logging.info("recv the data from " + str(addr))  
    p = Process(target=self.do\_recvdata, args=(recv\_data, addr))  
    p.start()  
    except Exception as e:  
    self.server\_socket.close()  
    print(e)  
    exit()  
      
    def do\_recvdata(self, recv\_data, addr):  
    data = self.trans\_request(recv\_data)  
    if data["method"] == "logout":  
    token = data["parm"]["token"]  
    status = self.logout(token)  
    return status  
    elif data["method"] == "send":  
    token = data["parm"]["token"]  
    message = data["parm"]["message"]  
    to\_id = data["parm"]["to\_id"]  
    from\_id = self.get\_id(token)  
    if from\_id == False:  
    return False  
    else:  
    status = self.send\_message(token, message, to\_id, from\_id)  
    response\_data = {  
    "status": status,  
    }  
      
    response\_data = json.dumps(response\_data).encode("utf-8")  
    self.server\_socket.sendto(response\_data, addr)  
    return True  
    elif data["method"] == "login" or data["method"] == "register":  
    if data["method"] == "login":  
    username = data["parm"]["username"]  
    password = data["parm"]["password"]  
    status = self.login(username, password, addr)  
    elif data["method"] == "register":  
    username = data["parm"]["username"]  
    password = data["parm"]["password"]  
    status = self.register(username, password, addr)  
    else:  
    status = False  
    if status != False:  
    token = status  
    send\_data = {  
    "status": "success",  
    "token": token  
    }  
    send\_data = json.dumps(send\_data).encode("utf-8")  
    self.server\_socket.sendto(send\_data, addr)  
    messages = self.get\_message\_not\_send(token)  
    for message in messages:  
    message\_id = message[0]  
    from\_id = message[2]  
    send\_data = {  
    "from\_id": from\_id,  
    "message": message[1].decode("utf-8")  
    }  
    send\_data = json.dumps(send\_data).encode("utf-8")  
    self.server\_socket.sendto(send\_data, addr)  
    self.update\_message(message\_id)  
    else:  
    send\_data = {  
    "status": "error"  
    }  
    send\_data = json.dumps(send\_data).encode("utf-8")  
    self.server\_socket.sendto(send\_data, addr)  
    return True  
      
    def get\_id(self, token):  
    *"""  
    get id by token* ***:param*** *token:* ***:return****:  
    """* return self.db.get\_user\_id(token)  
      
    def login(self, username, password, addr):  
    *"""  
    login by username and password* ***:param*** *username:* ***:param*** *password:* ***:param*** *addr:* ***:return****: token or FALSE  
    """* try:  
    token = self.db.get\_token(username, password)  
    self.db.login(token, addr)  
    return token  
    except:  
    return False  
      
    def register(self, username, password, addr):  
    *"""  
    register by username and password* ***:param*** *username:* ***:param*** *password:* ***:param*** *addr:* ***:return****: token or FALSE  
    """* try:  
    token = self.db.insert\_user(username, password)  
    self.db.login(token, addr)  
    return token  
    except:  
    return False  
      
    def logout(self, token):  
    *"""  
    logout user's status* ***:param*** *token:* ***:return****:  
    """* try:  
    self.db.logout(token)  
    return True  
    except:  
    return False  
      
    def trans\_request(self, data):  
    *"""  
    json loads the data which be recv* ***:param*** *data: bytes* ***:return****: dict  
    """* data = data.decode("utf-8")  
    return json.loads(data)  
      
    def get\_message\_not\_send(self, token):  
    *"""  
    get message by token* ***:param*** *token:* ***:return****: list  
    """* return self.db.get\_message(token)  
      
    def send\_message(self, token, message, to\_id, from\_id):  
    *"""  
    this function for send message* ***:param*** *token: string* ***:param*** *message: string* ***:param*** *to\_id: int* ***:param*** *from\_id: int* ***:return****: bool  
    """* is\_login, addr = self.db.get\_user\_status(to\_id)  
    send\_ids = []  
    not\_send\_ids = []  
    if is\_login == 1:  
    send\_data = {  
    "from\_id": from\_id,  
    "message": message,  
    }  
    send\_data = json.dumps(send\_data).encode("utf-8")  
    self.server\_socket.sendto(send\_data, addr)  
    send\_ids.append(to\_id)  
    else:  
    not\_send\_ids.append(to\_id)  
    return self.db.sent\_message(token, not\_send\_ids, send\_ids, message)  
      
    def update\_message(self, message\_id):  
    return self.db.update\_message(message\_id)  
     
     
   if \_\_name\_\_ == '\_\_main\_\_':  
    servers()

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
*"""  
\_\_title\_\_ = ''  
\_\_author\_\_ = 'ralph'  
\_\_mtime\_\_ = '2018/10/16'  
"""*import sqlite3  
import json, random  
import hashlib  
import logging  
  
logging.basicConfig(level=logging.INFO)  
  
  
class database\_init():  
 *"""  
 this class only for create database  
 """* def \_\_init\_\_(self):  
 self.init\_database()  
   
 def init\_database(self):  
 try:  
 database\_conn = sqlite3.connect('chatRoom.db')  
 logging.info("Opened database successfully")  
 database\_conn.execute('''CREATE TABLE users\_information(  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 username VARCHAR(128) NOT NULL,  
 password VARCHAR(128) NOT NULL,  
 token VARCHAR(128) NOT NULL,  
 friends TEXT,  
 is\_login INTEGER NOT NULL,  
 ip VARCHAR(128) NOT NULL  
 );  
 ''')  
 database\_conn.commit()  
 logging.info("user information table created successfully")  
 database\_conn.execute('''CREATE TABLE message(  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 message\_text TEXT,  
 to\_user INTEGER NOT NULL,  
 from\_user INTEGER NOT NULL,  
 has\_sent INTEGER NOT NULL  
 );  
 ''')  
 logging.info("user information table created successfully")  
 database\_conn.commit()  
 database\_conn.close()  
 logging.info('init the database success!')  
 except Exception as e:  
 logging.error("init the database error")  
  
  
class database():  
 *"""  
 this class is connect from server to database,  
 all of it depends on token  
 """* def \_\_init\_\_(self):  
 *"""  
 init database connect  
 """* try:  
 self.database\_conn = sqlite3.connect('chatRoom.db')  
 logging.info("connect database succsss")  
 except:  
 logging.error("connect database error")  
   
 def get\_token(self, username, password):  
 *"""  
 selct user's token from username and password* ***:param*** *username:* ***:param*** *password:* ***:return****: token or False  
 """* user = self.database\_conn.execute("SELECT username FROM users\_information WHERE username=?", [username])  
 if user.fetchall() == []:  
 return False  
 else:  
 password = hashlib.md5(password.encode("utf-8")).hexdigest()  
 token = self.database\_conn.execute("SELECT token FROM users\_information WHERE username=? AND password=?",  
 [username, password]).fetchall()[0][0]  
 return token  
   
 def insert\_user(self, username, password):  
 *"""  
 insert user from username and password* ***:param*** *username:* ***:param*** *password:* ***:return****: token or error  
 """* user = self.database\_conn.execute("SELECT username FROM users\_information WHERE username=?", [username])  
 if user.fetchall() == []:  
 friend = json.dumps([])  
 random\_str = str(int(random.random() \* 10000000000)) + username + str(int(random.random() \* 10000000000))  
 token = hashlib.md5(random\_str.encode('utf-8')).hexdigest()  
 password = hashlib.md5(password.encode("utf-8")).hexdigest()  
 ip = json.dumps(("0.0.0.0", 23333))  
 self.database\_conn.execute("INSERT INTO users\_information VALUES (NULL,?,?,?,?,?,?)",  
 [username, password, token, friend, 0, ip])  
   
 return token  
 else:  
 return False  
   
 def insert\_frients(self, token, friend\_id):  
 user\_id = self.database\_conn.execute("SELECT id FROM users\_information WHERE token=?", [token]).fetchall()  
 if user\_id == []:  
 return "error"  
 else:  
 friends = \  
 self.database\_conn.execute("SELECT friends FROM users\_information WHERE token=?", [token]).fetchall()[0][0]  
 friends = json.loads(friends)  
 friends.append(friend\_id)  
 friends = json.dumps(friends)  
 self.database\_conn.execute("UPDATE users\_information SET friends=? WHERE token=?", [friends, token])  
 self.database\_conn.commit()  
 return True  
   
 def get\_friends(self, token):  
 *"""  
 get friends by token* ***:param*** *token:* ***:return****: list of friends id or error  
 """* friends = self.database\_conn.execute("SELECT friends FROM users\_information WHERE token=?", [token]).fetchall()  
 if friends == []:  
 return False  
 else:  
 friends = json.loads(friends[0][0])  
 return friends  
   
 def login(self, token, ip):  
 *"""  
 update user's status about is\_login and ip* ***:param*** *token:* ***:param*** *ip:* ***:return****: status  
 """* user = self.database\_conn.execute("SELECT id from users\_information WHERE token=?", [token]).fetchall()  
 if user == []:  
 return False  
 else:  
 ip = json.dumps(ip)  
 self.database\_conn.execute("UPDATE users\_information SET is\_login=1,ip=? WHERE token=?", [ip, token])  
 self.database\_conn.commit()  
 return True  
   
 def logout(self, token):  
 *"""  
 update user's status about is\_login and ip* ***:param*** *token:* ***:return****:  
 """* user = self.database\_conn.execute("SELECT id from users\_information WHERE token=?", [token]).fetchall()  
 if user == []:  
 return False  
 else:  
 ip = json.dumps(("0.0.0.0", 23333))  
 self.database\_conn.execute("UPDATE users\_information SET is\_login=0,ip=? WHERE token=?", [ip, token])  
 self.database\_conn.commit()  
 return True  
   
 def get\_message(self, token):  
 *"""  
 get messages when you haven't recive* ***:param*** *token:* ***:return****:  
 """* user\_id = self.database\_conn.execute("SELECT id from users\_information WHERE token=?", [token]).fetchall()  
 if user\_id == []:  
 return False  
 else:  
 messages\_list = self.database\_conn.execute(  
 "SELECT id,message\_text,has\_sent,from\_user FROM message WHERE to\_user=?", user\_id[0]).fetchall()  
 messages = []  
 for message in messages\_list:  
 if message[2] == 0:  
 messages.append([message[0], message[1], message[3]])  
 return messages  
   
 def sent\_message(self, token, not\_send\_ids, send\_ids, message):  
 *"""  
 this function for user to send message to haven't login user* ***:param*** *token:* ***:param*** *from\_id:* ***:param*** *to\_ids:* ***:param*** *message:* ***:return****:  
 """* user\_id = self.database\_conn.execute("SELECT id FROM users\_information WHERE token=?", [token]).fetchall()  
 if user\_id == []:  
 return False  
 else:  
 user\_id = user\_id[0][0]  
 message = message.encode('utf-8')  
 for to\_id in not\_send\_ids:  
 self.database\_conn.execute("INSERT INTO message VALUES (NULL,?,?,?,?)", [message, to\_id, user\_id, 0])  
 for to\_id in send\_ids:  
 self.database\_conn.execute("INSERT INTO message VALUES (NULL,?,?,?,?)", [message, to\_id, user\_id, 1])  
 self.database\_conn.commit()  
 return True  
   
 def update\_message(self, message\_id):  
 try:  
 self.database\_conn.execute("UPDATE message SET has\_sent=1 WHERE id=?", [message\_id])  
 self.database\_conn.commit()  
 return True  
 except:  
 return False  
   
 def get\_all\_login\_user(self):  
 users = self.database\_conn.execute("SELECT id,ip FROM users\_information WHERE is\_login=1").fetchall()  
 return list(users)  
   
 def get\_user\_status(self, user\_id):  
 users = self.database\_conn.execute("SELECT is\_login,ip FROM users\_information WHERE id=?", [user\_id]).fetchall()  
   
 if users == []:  
 return False  
 else:  
 is\_login = users[0][0]  
 addr = tuple(json.loads(users[0][1]))  
 return is\_login, addr  
   
 def get\_user\_id(self, token):  
 users = self.database\_conn.execute("SELECT id FROM users\_information WHERE token=?", [token]).fetchall()  
 if users == []:  
 return False  
 else:  
 return users[0][0]

1. **客户端源码**

import socket  
import json  
from PyQt5.QtWidgets import QDialog  
  
  
class client():  
 def \_\_init\_\_(self, socket\_connect=None):  
 self.socket\_connect = socket.socket(socket.AF\_INET,  
 socket.SOCK\_DGRAM) if socket\_connect == None else socket\_connect  
   
 self.server\_addr = ("127.0.0.1", 23333)  
   
 self.token = "0"  
 self.id = -1  
 # self.socket\_connect.connect(self.server\_addr)  
   
 def login(self, username, password):  
 data = {  
 "method": "login",  
 "parm": {  
 "username": username,  
 "password": password  
 }  
 }  
   
 data = json.dumps(data).encode("utf-8")  
 self.socket\_connect.sendto(data, self.server\_addr)  
 r = self.socket\_connect.recv(1024).decode("utf-8")  
 response = json.loads(r)  
 status = response['status']  
 if status == "success":  
 self.token = response["token"]  
 self.id = self  
 return True  
 else:  
 return False  
   
 def register(self, username, password):  
 data = {  
 "method": "register",  
 "parm": {  
 "username": username,  
 "password": password  
 }  
 }  
   
 data = json.dumps(data).encode("utf-8")  
 self.socket\_connect.sendto(data, self.server\_addr)  
 r = self.socket\_connect.recv(1024).decode("utf-8")  
 response = json.loads(r)  
 status = response['status']  
 if status == "success":  
 self.token = response["token"]  
 return True  
 else:  
 return False  
   
 def send\_message(self, message, token, to\_id):  
 data = {  
 "method": "send",  
 "parm": {  
 "token": self.token,  
 "message": message,  
 "to\_id": to\_id  
 }  
 }  
   
 data = json.dumps(data).encode("utf-8")  
 self.socket\_connect.sendto(data, self.server\_addr)  
 return True  
   
 def recv\_message(self):  
 response = self.socket\_connect.recv(1024).decode("utf-8")  
 response = json.loads(response)  
 if "message" in response:  
 return response  
 else:  
 return False  
   
 def log\_out(self):  
 data = {  
 "method": "logout",  
 "parm": {  
 "token": self.token  
 }  
 }  
 data = json.dumps(data).encode("utf-8")  
 self.socket\_connect.sendto(data, self.server\_addr)

import sys  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
from PyQt5.QtGui import \*  
from client import client  
  
from multiprocessing import Process  
  
import chat, login  
  
class MainWindow(QMainWindow, chat.Ui\_MainWindow):  
 *"""  
 the class of main windows  
 user must login or register success  
 """* def \_\_init\_\_(self, parents=None):  
 super(MainWindow, self).\_\_init\_\_(parents)  
 self.setupUi(self)  
 self.pushButton.clicked.connect(self.send\_message)  
   
   
 def set\_connect(self, client):  
 self.client = client  
   
 def send\_message(self):  
 to\_id = self.textEdit\_2.toPlainText()  
 message = self.textEdit\_3.toPlainText()  
 if message != "" and to\_id != "":  
 try:  
 to\_id = int(to\_id)  
 self.client.send\_message(message, self.client.token, to\_id)  
 show\_text = "To {0} : {1}".format(str(to\_id),message)  
 self.listWidget.addItem(show\_text)  
   
 except:  
 self.textEdit\_3.setText("error to send message")  
 else:  
 self.textEdit\_3.setText("error to send")  
   
  
 def resvmessage(self,recv\_data):  
 message = recv\_data["message"]  
 from\_id = recv\_data["from\_id"]  
 show\_text = "From {0} : {1}".format(str(from\_id),message)  
 self.listWidget.addItem(show\_text)  
   
  
 def closeEvent(self, event):  
 *"""  
 重写closeEvent方法，实现dialog窗体关闭时执行一些代码* ***:param*** *event: close()触发的事件* ***:return****: None  
 """* reply = QMessageBox.question(self,'本程序',"是否要退出程序？",QMessageBox.Yes | QMessageBox.No,QMessageBox.No)  
 if reply == QMessageBox.Yes:  
 self.client.log\_out()  
 event.accept()  
 else:  
 event.ignore()  
   
   
class logindialog(QDialog, login.Ui\_Dialog):  
 def \_\_init\_\_(self, parents=None):  
 super(logindialog, self).\_\_init\_\_(parents)  
 self.setupUi(self)  
   
 self.pushButton.clicked.connect(self.login)  
 self.pushButton\_2.clicked.connect(self.register)  
   
 def set\_connect(self, client):  
 self.client = client  
   
 def login(self):  
 username = self.lineEdit.text()  
 password = self.lineEdit\_2.text()  
 if username == "" or password == "":  
 self.label\_4.setText("login error")  
 else:  
 status = self.client.login(username,password)  
 if status == True:  
 self.accept()  
 else:  
 self.label\_4.setText("login error")  
   
 def register(self):  
 username = self.lineEdit.text()  
 password = self.lineEdit.text()  
 if username == "" or password == "":  
 self.label\_4.setText("register error")  
 else:  
 status = self.client.login(username,password)  
 if status == True:  
 self.accept()  
 else:  
 self.label\_4.setText("register error")  
   
   
class WorkTread(QThread):  
 recv\_data = pyqtSignal(object)  
   
 def \_\_init\_\_(self,conn):  
 super(WorkTread,self).\_\_init\_\_()  
 self.c\_conn = conn  
   
 def run(self):  
 while True:  
 response = self.c\_conn.recv\_message()  
 if response == False:  
 continue  
 else:  
 self.recv\_data.emit(response)  
   
   
   
  
def main():  
 app = QApplication(sys.argv)  
 c\_conn = client()  
 dialog = logindialog()  
 dialog.set\_connect(c\_conn)  
 if dialog.exec\_() == QDialog.Accepted:  
 the\_window = MainWindow()  
 the\_window.set\_connect(c\_conn)  
 recv\_data = WorkTread(c\_conn)  
 recv\_data.recv\_data.connect(the\_window.resvmessage)  
 recv\_data.start()  
 the\_window.show()  
 sys.exit(app.exec\_())  
   
   
   
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **程序测试方法及测试结果记录（不能光截图，要有相应的文字说明）**
2. **测试方法**

**本地化测试以及分模块测试**

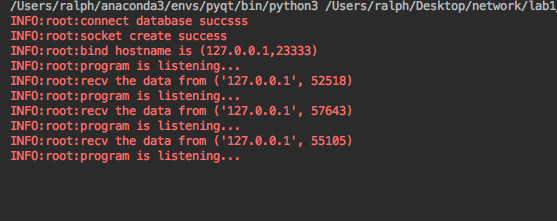
1. **测试流程**
2. 测试数据库处理

根据数据库模块对数据库进行测试。

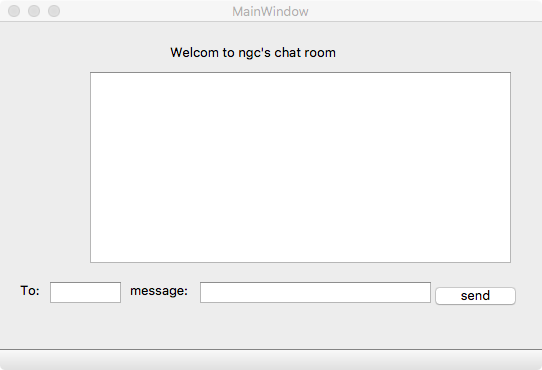
1. 本地测试服务端与客户端相互交互测试。

首先登陆三个账户/注册三个函数

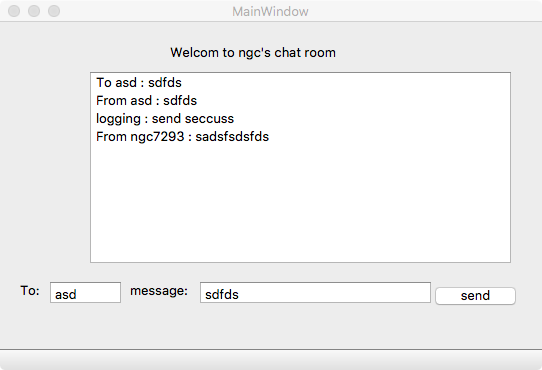
服务端截图

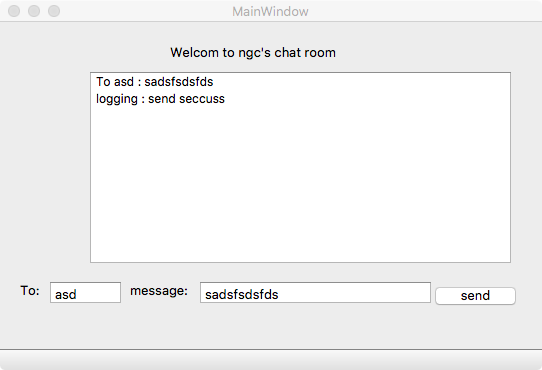


客户端截图

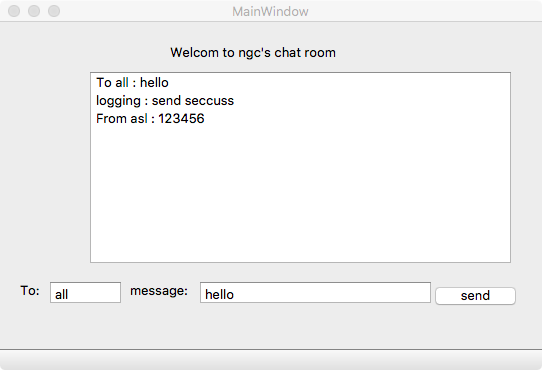


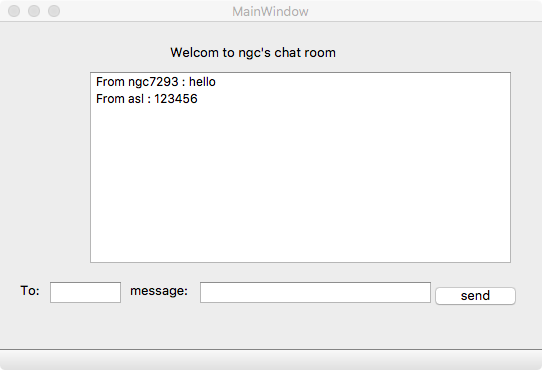
聊天测试

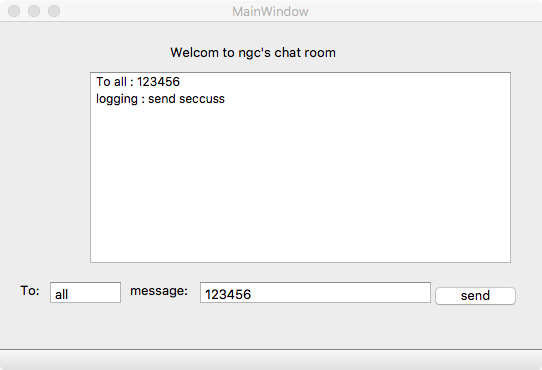




群发测试



****

****

1. **实验分析总结及心得**
2. **关于pyqt在客户端事件主循环中不可以用process直接创建新的子进程**

**首先，我们在客户端开发的时候注意到一个问题，当我们在主事件循环中开启一个进程来循环接收消息并，但是运行时会报错，大致原因是在qt主事件中不可以重新创建进程，我们需要创建一个Qthread的子类，将socket对象传入，然后将其循环接收消息，当接受到时给用户主事件emit信号，并由主事件进行处理。**

1. **关于退出账号的操作**

**考虑过一些udp如何保持连接，包括心跳包以及类tcp连接的方式，但是考虑到对网络的要求以及作为作业的简单性，只为系统留出了接口，包括用户friends的添加删除，拉黑名单，离线消息的收发，用户是否在线等，而用户退出则没有采用心跳包而是当用户ui退出后将整个连接关闭，server端直接关闭子进程，另外在server端维护一个在线的列表，当发送不出消息或服务端子进程关闭时将移除该用户，在登陆或注册时会将其id，username以及address记录下来。**

1. **服务端对于各个操作的封装**
   1. **数据库层面的封装**

**在项目中，对数据库的所有操作进行了封装包括将登陆注册收发信息等，将其接口暴露给server，另外对数据库的初始化进行封装，使得可以一键创建数据库，简化了创建数据库的流程。**

* 1. **server端收发消息的封装**

**在项目中，server端需要与client交互，与database交互，是整个项目的核心，无论是群发还是私聊，我们都采用的是简化client端而将消息发给server由server 进行转发。在server端，封装了所有的操作，便于以后进行重构以及拓展，目前只是保持了最基本的要求，保证了拓展性。**

（结合所学知识对实验过程中观察到的实验结果进行分析总结，以便加深对知识的理解，并总结通过实验学到的知识或技术）