实验报告

多用户实时在线聊天系统

题目

设计并实现多用户实时在线聊天系统。用户可以该聊天系统进行一对一或者一对多的聊天，保证聊天的性能，同时保证聊天内容的一致性。

实验要求

- 支持一对一聊天;
- 支持聊天室群聊;
- 能够满足实时性要求如响应时间控制在 10ms 以内;
- 通信方式采用 RPC;
- 支持分布式系统一致性;
- 具备一定的失效容错措施;
- 需进行性能测试;

环境配置

python3 grpcio grpcio-tools protobuf tkinter

具体实现

$server - client - proto$

- proto 文件
  - 这些在上一次的rpc作业中已经大规模地使用过，所以实现起来较简单

```
syntax = "proto3";
package grpc;
message Empty {}

message Note {
    string name = 1;
    string message = 2;
}

service ChatServer {
    // This bi-directional stream makes it possible to send and
    //receive Notes between 2 persons
    rpc ChatStream (Empty) returns (stream Note);
    rpc SendNote (Note) returns (Empty);
}
```

- server
  - 调用tkinter选择群聊或一对一聊天

```
if __name__ == '__main__':
top = tkinter.Tk()
top.title('SERVER')
top.geometry('500x300')

text = tkinter.Label(top, text='欢迎来到GZY实时聊天系统！')
A = tkinter.Button(top, text="一对一聊天", command=one_to_one)
B = tkinter.Button(top, text="一对多群聊", command=one_to_all)
text.pack(expand='yes')
A.pack(expand='yes')
B.pack(expand='yes')

top.mainloop()
openserver(chater)
```

  - 同时以防他人加入设置最大chater人数

```
def one_to_one():
global chater
chater = 3
A.configure(state='disabled')
tkinter.messagebox.showinfo('Tips','请勿开启超过2个客户端！\n否则将视为聊天
被窥窃，程序保护性崩溃！\n')
top.destroy()

def one_to_all():
    global chater
    chater = 10
```

```
    B.configure(state='disabled')
    tkinter.messagebox.showinfo('Tips','请勿开启超过10个客户端！\n否则将视为聊天被窥窃，终止程序！\n')
    top.destroy()
```

- 启动服务器

```
def openserver(chater):
port = 11912  # a random port for the server to run on
# the workers is like the amount of threads that can be opened at the same time, when there are 10 clients connected
# then no more clients able to connect to the server.

server = grpc.server(futures.ThreadPoolExecutor(
    max_workers=chater))  # create a gRPC server
rpc.add_ChatServerServicer_to_server(
    ChatServer(), server)  # register the server to gRPC
# gRPC basically manages all the threading and server responding logic, which is perfect!
print('Starting server. Listening...')
server.add_insecure_port('[::]:' + str(port))
server.start()
# Server starts in background (in another thread) so keep waiting
# if we don't wait here the main thread will end, which will end all the child threads, and thus the threads
# from the server won't continue to work and stop the server
while True:
    time.sleep(64 * 64 * 100)
```

- client
  - 调用tkinter设置界面

```
def __setup_ui(self):
self.chat_list = Text()
self.chat_list.pack(side=TOP)
self.lbl_username = Label(self.window, text=self.username)
self.lbl_username.pack(side=LEFT)
self.entry_message = Entry(self.window, bd=5)
self.entry_message.bind('<Return>', self.send_message)
self.entry_message.focus()
self.entry_message.pack(side=BOTTOM)
```

  - grpc连接服务器

```
def __init__(self, u: str, window):
# the frame to put ui components on
```

```
self.window = window
self.username = u
# create a gRPC channel + stub
channel = grpc.insecure_channel(address + ':' + str(port))
self.conn = rpc.ChatServerStub(channel)
# create new listening thread for when new message streams come in
threading.Thread(target=self.__listen_for_messages, daemon=True).start()
self.__setup_ui()
self.window.mainloop()
```

- 发送信息

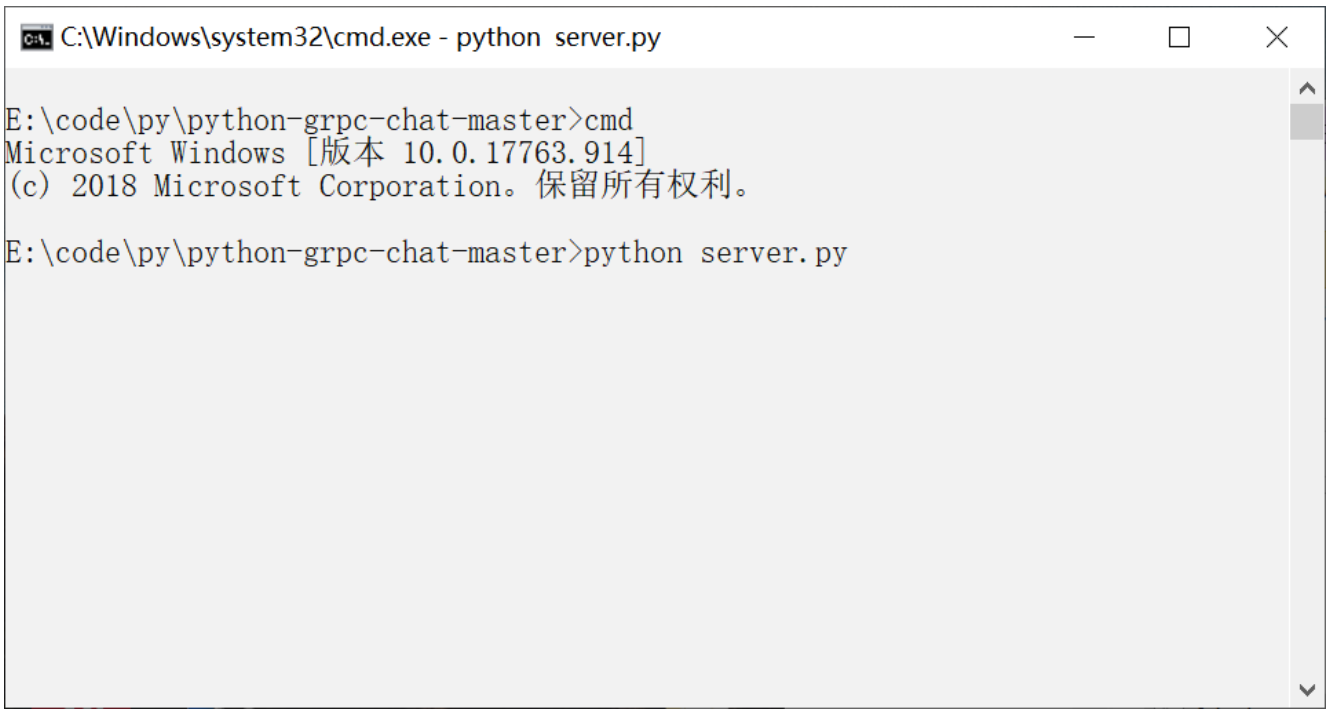
```
def send_message(self, event):
"""
This method is called when user enters something into the textbox
"""
message = self.entry_message.get()  # retrieve message from the UI
if message is not '':
    n = chat.Note()  # create protobug message (called Note)
    n.name = self.username  # set the username
    n.message = message  # set the actual message of the note
    print("S[{}] {}".format(n.name, n.message))  # debugging statement
    self.conn.SendNote(n)  # send the Note to the server
```

功能测试

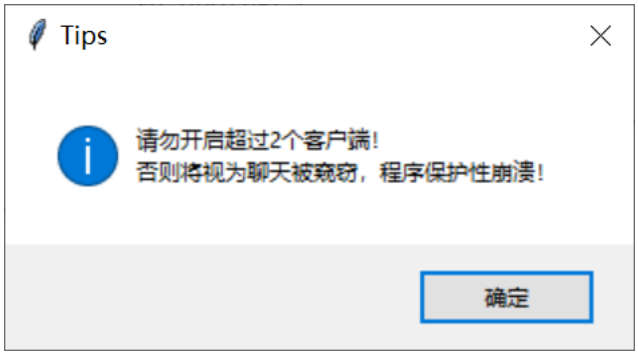
首先运行

```
python server.py
```

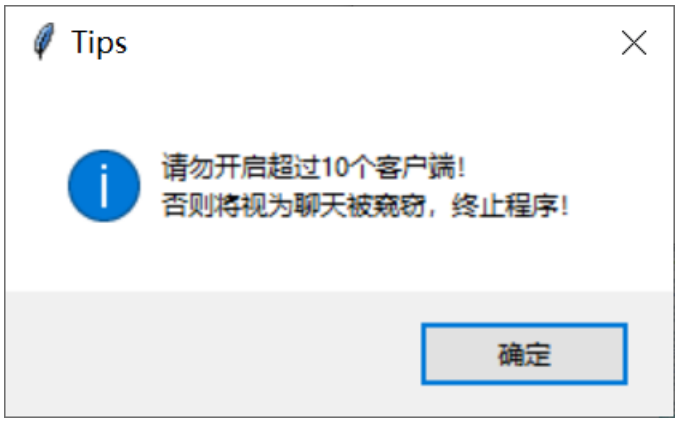
如下图：



选择一对一聊天，则会跳出窗口如下：

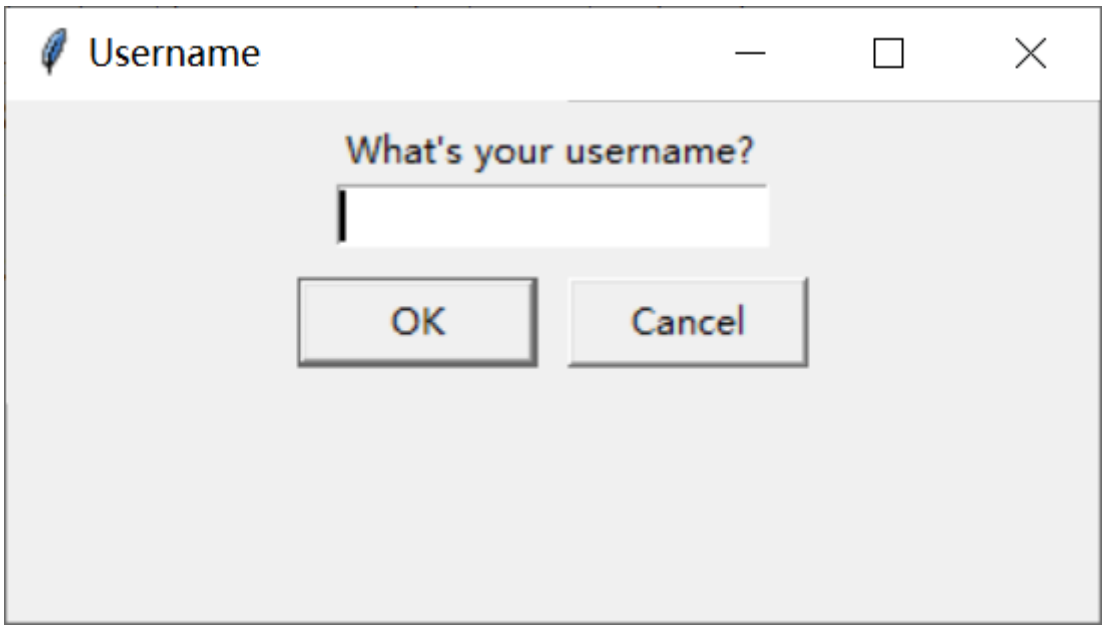


选择群聊，则会跳出窗口如下：

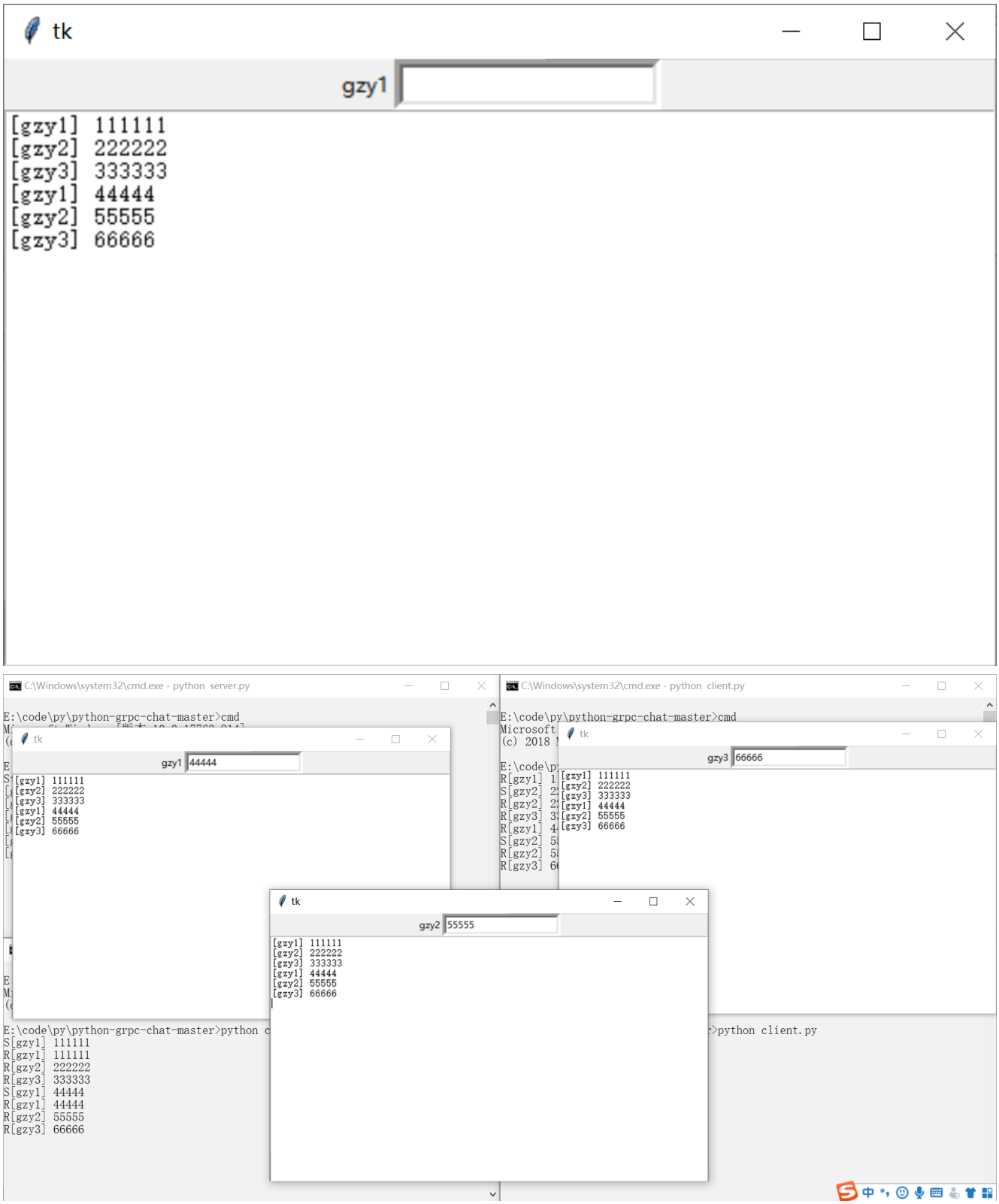


对程序的失效容错做出提示。

接着选择群聊，建立多个客户端，进行聊天，截图如下：
弹出输入用户名窗口：



输入后进入聊天室，通过上方的输入框以及回车即可输入并发送信息，进行聊天，经测试，聊天的延迟不高，符号实时性。



选择一对一聊天的结果是差不多的，只不过是在使用第三个client连接server时会发生超过max_workers进而，线程池满，拒绝访问而退出程序。也算是一种保护，其他的聊天画面与群聊一致，此处就不多做展示。

经以上测试，实现的功能基本符合实验要求。

总结

这是我根据之前接触grpc编程以及TA给出参考的代码做的第二个项目，聊天系统。利用tkinter库设计了自己的界面，然后使用按钮来选择聊天形式。相比文件中给出的聊天借鉴的项

目来看，虽然简陋了一些，但却易于实现，用的也是之前的学过的grpc，写起来相对顺利。相比参考文件的代码实现，自己设计的容错措施以及界面还是有独特的想法。