EG - 101

**Socket Programming Project**

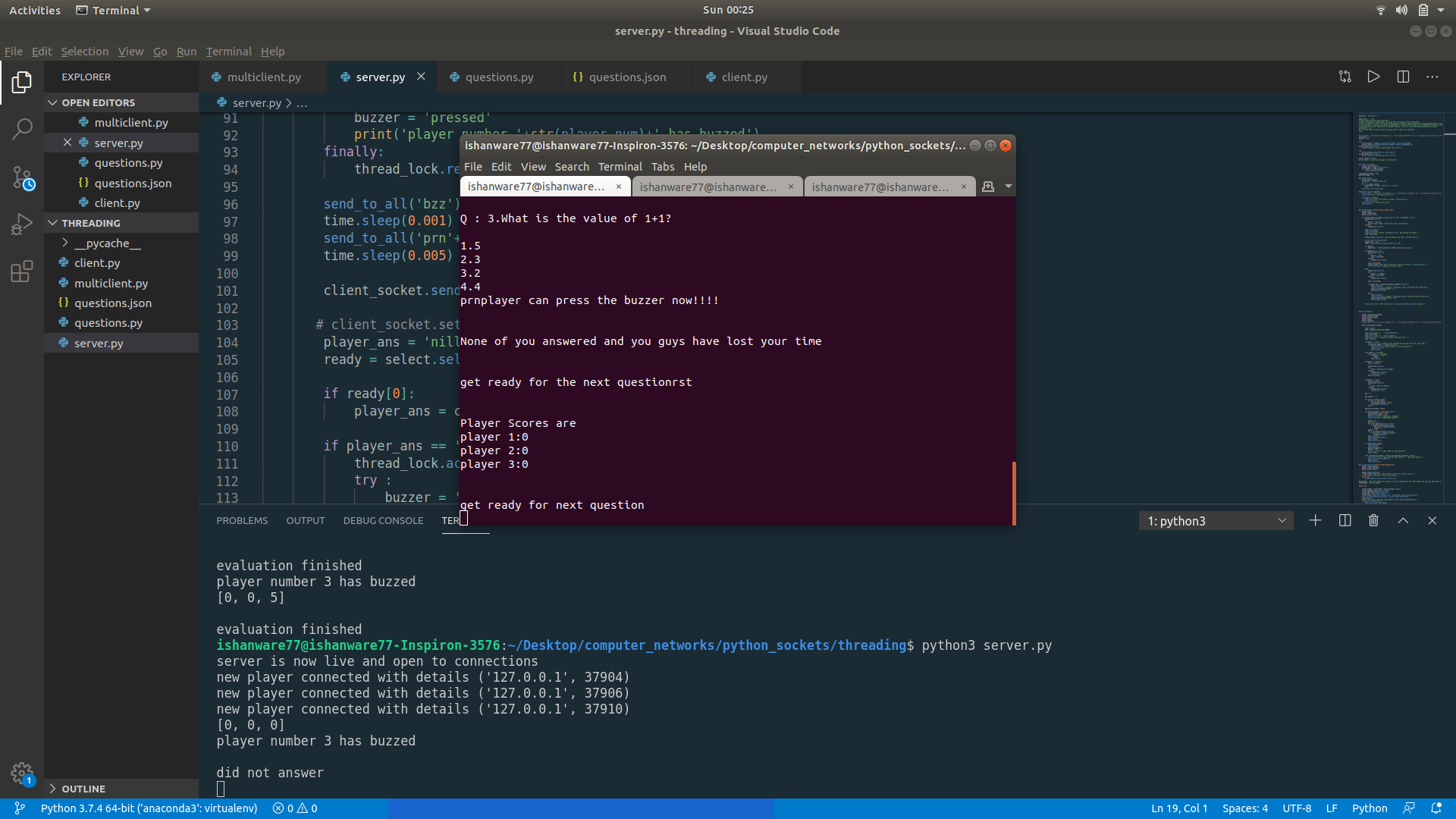
The Game Show

horizontal line

# 

# Introduction

This Project is about a game show where a host conducts the show, asks questions and participants/players provide answers. Let us say there are three participants. He randomly chooses one of the questions (making sure it is not a repeat from a list of questions) and sends it to all three players. The player receives the question, thinks about the answer for a while and presses the buzzer. There is a timer for 10 seconds for the buzzer to be pressed. Otherwise, the host moves on to the next question. The first one to press the buzzer is given a chance to provide the answer within 10 seconds. If the answer is correct, he is given 1 point, otherwise -0.5. The host then proceeds with the next question. The game stops when any player gets 5 points and that player is declared the winner. So We can think of the host to be the server and all the three individual players to be the clients connected to the server.



## 

## Assumptions

1. To complete the game show, there is a possibility that the no player reaches the maximum score that is 5, even after the questions are finished. In such a case the person with the maximum score would win the game.

In all the cases as soon as a particular user reaches the max score before the end of the questions, the player would be declared as the winner.

1. There is no other application running on port number 12345 on the pc/laptop being used as the server.
2. Both the server and client systems should have Python3 installed.

### Description of the Code

**Multithreading**

In this application the server can be thought of as the host of the game show, and each individual client as the player of the game show. Socket programming is used to connect these clients to the server, where the client only sends and accepts data. The server also sends and receives data however it analyses it as well, and then calls for the next step based on the data received from the client.

The project uses the concept of multithreading, which is used to run multiple threads simultaneously in a single process. This method allows us to run different tasks at the same time. Threads allow us to control which process to execute and run parallely. To understand this better I will explain the project.

In the server.py file the main\_thread function is the function which takes care of the questions which are sent to all the players connected to the server. However, to allow a task such as buzzer handling, we need to handle each client on a unique thread. So that after one of the clients has pressed the buzzer we can lock that client’s unique thread preventing other clients from pressing the buzzer . It also disallows the main\_thread from posting the next question until the client has answered.

This is done by locking the thread of the player who has pressed the buzzer(disallowing any other player from accessing the variable) and then changing the state of the global buzzer variable to ‘pressed’ disallowing the main\_thread from continuing after the 10 second window of pressing the buzzer.

Since each client is handled using a unique thread, the functions running on their thread manages or handles the responses received from the client at the same time as the main\_thread is still running. This unique thread checks if the answer is correct or incorrect and then updates the global scores list accordingly. This is the main idea the project has been built upon.

After each evaluation by the threads, main\_thread() checks if max score has reached, main\_thread() also keep track of the question number.

**Socket Programming and client file**

Further ,Socket programming is used to establish the connection between the host and the player, i.e, the server and the client. I have used TCP , to establish the connection by specifying socket.STREAM while creating the server and client socket respectively. The server socket is then binded to the ‘local host’ ip address(127.0.0.1), which the client specifies while connecting.

TCP is used to ensure reliable data transfer, so that we can ensure that all data transmitted is received by the destination system(which can be server or client).

For eg: we need to ensure that ‘ans’ sent from the client is not lost and reaches the server safely.

Server accepts connections on port number 12345. Inorder to connect to the game host server, the client socket is connected to the server on the same port 12345 on which the server host is running. This prevents it from connecting to any other application running on the server.

The Program uses the select module, this monitors sockets and other input channels(such as keyboard) continuously and then decides which input channel is functional. With the help of the select module, the code for the client side literally reduced to half.

Since the operating system network layer was monitoring the input channels. By identifying the input channel it could decide, either to take input from the client or print a message from the server.

**Flagging**

I have used the flagging method on the client side , which helps us control the kind of input that the client receives from the server, and the kind of input the user sends to the server.

For example, a ‘can’ message sent by the server allows all user to answer, by changing their response state = 2(this response state allows data to be sent to server), the ‘bzz’ msg prevents all users from answering(sent after buzzer is pressed ,response\_state =0), ‘ans’ message sent to buzzer presser, changes his response state = 1, allowing only him to send the message.

Some flag notations i have used:

bzz --> buzzer has been pressed.

ans --> Send answer now

rst --> Reset responseState

can --> Receive and print options

kll --> Terminate(after max score reached)

Finally, the python modules i am using and the purpose,

**Socket Module** - To create and control the socket objects used to establish connections between client and server.

**Select Module** - To identify the input stream on the client side and create a 10s window after pressing the buzzer on the server’s side, to limit the time until which the user can respond.

**Threading Module** -To create thread object and the global lock object and handling each user on a unique thread, this allows us to execute processes parallely

**Time Module** - To manage the time delays and timeouts, such as the time delay of 10sec on the main\_thread, while other threads run parallely to check for buzzer presses in the meantime.

**Random module-** To shuffle questions

**Some Instructions to run the project:**

**Make sure all the three python file which are server.py ,client.py and questions.py**

**Are in the same folder.**

**Run the server file**

**Using the command**

python3 server.py

**Run the client file**

**Using the command**

Python3 client.py

**In order to press the buzzer you only need to press enter.**

**Only send the option number while giving the ans.**

**Don't exit the or press ctrl+c on either side until the game ends.**