Intro to Nets 2021 Hackathon Assignment - Quick Maths ②+②-①=③👑

Version 1.0

Introduction

Your objective is to write a client-server application which will implement a fast-paced math game. The players in this game get a random math question and are supposed to answer it correctly as fast as possible.
Each team in the course will write both a client application and a server application, and everybody's clients and servers are expected to work together with full compatibility.

Example Run

1. Team Mystic starts their server. The server prints out "`Server started, listening on IP address 172.1.0.4`" and starts automatically sending out "offer" announcements via UDP broadcast once every second.
2. Team Valor starts their server, which prints out "`Server started, listening on IP address 172.1.0.88`" and also starts automatically sending out "offer" announcements over UDP once every second.
3. Teams Instinct, Rocket, Beitar and Katamon start their clients. The clients all print out "`Client started, listening for offer requests...`". All of the clients get the Mystic announcement first, and print out ""`Received offer from 172.1.0.4, attempting to connect...`"
4. Each client connects to the Team Mystic server over TCP. After the connection succeeds each client sends the team name over the TCP connection, followed by a line break ('\n')
5. The server allows the first two users to connect, but rejects the third request
6. 10 seconds after the second user joins, the game begins - the server sends a welcome message to all of the clients with the names of the teams, followed by a randomly-generated simple math problem. The answer to the math question should be exactly a single digit, for example:

```
Welcome to Quick Maths.
Player 1: Instinct
Player 2: Rocket
==
Please answer the following question as fast as you can:
```

```
How much is 2+2?
```

7. Each client receives this message over TCP and prints it to the screen. Now every time the user types in a key the client sends it to the server over TCP
8. Each time the server receives a key over TCP, it checks the correctness of the answer:

- If the answer is correct, the team wins immediately
- If the answer is incorrect, the other team wins immediately
- If nobody answers after 10 seconds, the game finishes with a draw

9. After the score of the game is decided, the server sends a summary message to both players, for example:

```
Game over!
The correct answer was 4!

Congratulations to the winner: Instinct
```

10. The server closes the TCP connection, prints "`Game over, sending out offer requests...`" and goes back to sending offer messages once a second
11. The clients print "`Server disconnected, listening for offer requests...`" and go back to waiting for offer messages

Suggested Architecture

The client is a single-threaded app, which has three states:

- Looking for a server. You leave this state when you get an offer message.
- Connecting to a server. You leave this state when you successfully connect using TCP
- Game mode - collect characters from the keyboard and send them over TCP. collect data from the network and print it on screen.

Note that in game mode the client responds to two events - both keyboard presses and data coming in over TCP. Think about how you can do this.

The server is multi-threaded since it has to manage multiple clients. It has two states:

- Waiting for clients - sending out offer messages and responding to request messages and new TCP connections. You leave this state after two users join the game.
- Game mode - collect characters from the network and decide the winner. You leave this state after 10 seconds, or after the first user tries to answer

Packet Formats

- Servers broadcast their announcements with destination port 13117 using UDP. There is one packet format used for all UDP communications:
    - Magic cookie (4 bytes): 0xabcddcba. The message is rejected if it doesn't start with this cookie
    - Message type (1 byte): 0x2 for offer. No other message types are supported.
    - Server port (2 bytes): The port on the server that the client is supposed to connect to over TCP (the IP address of the server is the same for the UDP and TCP connections, so it doesn't need to be sent).
- The data over TCP has no packet format. After connecting, the client sends the predefined team name to the server, followed by a line break ('\n'). After that, the client simply prints anything it gets from the server onscreen, and sends anything it gets from the keyboard to the server.

.

Tips and Guidelines

- Please pick a creative name for your team - there will be a contest.
- Both server and client applications are supposed to run forever, until you quit them manually. You will not get full points if your program quits or crashes, even if it's because of some network-related problems.
- The server does not have to listen on any particular port over TCP, since this is part of the offer message.
- **Do not use busy-waiting (e.g. while-continue loops), or your program will not be allowed to run on the cloud server.** As a general guideline, if your program consumes more than 1% CPU on your own computers, you're probably doing something wrong.
- Think about what you should do if things fail - messages are corrupted, the server does not respond, the clients hang up, etc. Your error handling code will be tested.
- If you try to run two clients on the same computer, they won't be able to listen on the same UDP port unless you set the SO_REUSEPORT option when you open the socket, like this:

```
>>> s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

>>> s.bind(('',13117))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 98] Address already in use

>>> s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
>>> s.bind(('',13117))
```

- The assignment will be written and tested on the hackathon lab machines, which you already experimented with during the first assignment. It's highly recommended that you install Visual Studio Code with Remote Development, which will allow you to use your own computers to do the coding, while running and debugging the code remotely on the hackathon computers over ssh.

- The assignment will be written in Python 3.x. You can use standard sockets or the scapy package for network programming. The struct.pack functions can help you encode and decode data from the UDP message.
- Please set up a git repository for your own code and make sure to commit regularly, since the file system on the hackathon computers is unstable. Visual Studio Code has git support so it's quite easy.
- The hackathon computers have two virtual networks. The dev network (eth1, 172.1.0/24) is used for development, and the test network (eth2, 172.99.0/24) will be used by the course staff to test your work. Please make sure you listen and send broadcasts only on the dev network unless you're being graded. Make sure your code has an easy way to choose which interface it works on. You can find your own IP address on each network by calling the scapy functions get_if_addr('eth1') or get_if_addr('eth2'). If you listen on "localhost" (127.0.0.1) you will only get messages from your own computer, and not from the network. This might be useful in the early stages of coding.
- The Va'adim will be organizing some fun activities during the Hackathon, including a contest with prizes. Please follow their own announcements about this.

To Get Full Points on Your Assignment

- Work with any client and any server
- Write high-quality code (see below)
- Have proper error handling for invalid inputs from the user and from the network, including timeouts
- Bonus: use ANSI color to make your output fun to read
- Bonus: collect and print interesting statistics when the game finishes (best team ever to play on this server, most commonly typed character, anything else you can think of…)

Code Quality Expectations

How to Submit

Please submit to the Moodle a link to your github repository (make sure it's not private). You can commit as much as you want, but only the last commit before the deadline will be considered.

Static Quality

- Code has proper layout, and meaningful function and variable names
- Code has comments and documentation for functions and major code blocks
- No hard-coded constants inside code, especially IP addresses or ports

Dynamic Quality

- Return values of functions are checked
- Exceptions are handled properly
- No busy-waiting!
- Network code is isolated to a single network

Source control

- Code is hosted in a github repository
- Commits were made by all members of the team
- Proper use of commit messages and branches

Good luck!