Task 2 [5 points]

`TCP`, `multithreading`

In this task, you need to write client-server app using TCP sockets and multithreading.

The server will notify all connected clients about names of their peers. Each client would print list of names sent by server.

You need to produce two files: client.py and server.py. Each of them is going to be started in following way:

```
python3 server.py port
python3 client.py server_ip:server_port NAME1
python3 client.py server_ip:server_port NAME2
python3 client.py server_ip:server_port NAME3
```

It is guaranteed that all clients are going to have unique names. Names are guaranteed to be no longer than 20 bytes. Names can consist only of letters and digits: `[a-zA-Z0-9]+`.

Server

Has 1 command-line argument:

1. `port` - port server is running on.

Example:

```
python3 server.py 5555
```

Server specification:

- When started, server reads `port` from commandline arguments, binds the TCP sockets and waits for the clients.
- Server should remember names of all currently connected clients.
- You can use list or dictionary for that purpose.
- Every time a single client connects or disconnects from the server, server must send to its clients a message.
- Message must be the list of client names sorted alphabetically and joined by space, followed by newline. For example, if server has three clients with names: bob, chris, alice, then the message to all its clients would be `"alice bob chris\n"`.
- When user hits Ctrl+C (`KeyboardInterrupt`), server should close client connections, gracefully shutdown threads and terminate.

Note: In order to handle several clients simulatenously, you need to use `threading.Thread` to handle each connection.

Note: When socket is closed, `sock.recv(BUF_SIZE)` returns empty string.

Client

Has 2 command-line arguments:

1. `server_ip:server_port` - ip address and port of the server.
2. `name` - client's name.

Example:

```
python3 client.py 127.0.0.1:5555 Bob
```

Client specification:

- When started, client reads `server_ip:server_port` and `name` from commandline arguments (`sys.argv`).
- Connects to the server over TCP and sends its `name` (once).
- After that, client continuously receives text lines from server and prints them one after another in the terminal.
- Client does not need to read anything from the terminal.
- When user hits Ctrl+C (`KeyboardInterrupt`), client should close the socket and terminate.

Submission

Put 2 files:

- client.py
- server.py

In directory `Task2`

Example

Run server:

```
python3 server.py 5555
```

Run clients:

```
python3 client.py 127.0.0.1:5555 greg
```

```
python3 client.py 127.0.0.1:5555 bob
```

```
python3 client.py 127.0.0.1:5555 vlad
```

First client will print:

```
greg
bob greg
bob greg vlad
```

Second client will print:

```
bob greg
bob greg vlad
```

Third client will print:

```
bob greg vlad
```

If we terminate second client, then both remaining clients will print:

```
greg vlad
```

Task 1 [5 points]

gRPC

In this task, you need to create a server and a client using grpc.

Server

Your server is a remote queue.

It has 2 command line arguments:

1. `port` - port server is running on.
2. `size` - maxsize of the queue.

Example:

```
python3 server.py 5555 10
```

And it has four rpc functions:

1. put(item) - adds a new element (item) to the end of the queue if the queue is not full.
2. peek() - returns the first element from the queue but does not remove it. Should be able to handle empty queue.
3. pop() - removes the first element from the queue and returns it. Should be able to handle empty queue.
4. size() - returns the current number of the elements in the queue.

Properties of the server:

- The queue state is the same for all clients.
- Outputs are not required.
- Any data structure can be used to implement the internal structure of the server.

Client

Your client must continuously prompt for user input (stops on Ctrl+C (`KeyboardInterrupt`)).

It has 1 command line argument:

1. `server_ip:server_port` - ip address and port of the server.

Example:

```
python3 client.py 127.0.0.1:5555
```

And be able to hande following commands from user:

1. put item - puts `item` to the remote queue. `item` is a string without spaces. Print the result of operation (True if the item was added, False if the queue was full).
2. peek - prints the first element in the queue. If the queue is empty, prints "Queue is empty".
3. pop - prints the first element from the queue and removes it. If the queue is empty, prints "Queue is empty".
4. size - prints the current number of the elements in the queue.

Submission

Put 3 files:

- server.py
- client.py
- queue.proto

In directory `Task1`

Example

Run server:

```
python3 server.py 5555 3
```

Run clients:

```
python3 client.py 127.0.0.1:5555
```

```
python3 client.py 127.0.0.1:5555
```

Output from a single client:

```
> put A
True
> put B
True
> put C
True
> put D
False       // Queue is full (size is 3)
> size
```

```
3
> peek
A
> size
3
> pop
A
> pop
B
> size
1
> pop
C
> pop
None        // Queue is empty
> peek
None        // Queue is empty
> ^CShutting down
```