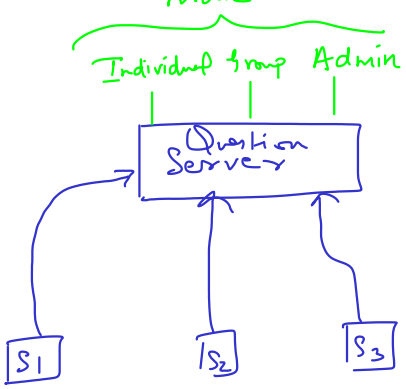
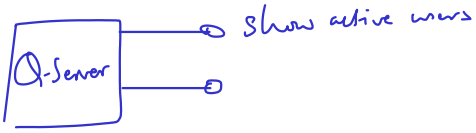
Online Q-A System with Individual / Group Quiz

Objective:

- To learn protocol design
- To learn to use AWS free tier as a relay server
- To learn socket programming
- To use threads to handle concurrent requests.



Topic:
Question:



- a user connects to the Question Server

#> Welcome <user-id> to Online Quiz on OS.

Select mode:

- press I for individual mode
- press G for group mode
- press A for admin mode

#> I

#> Ok. Pick a topic from the following:

1 Threads
2 Scheduling
3 Memory management

} populated in admin mode

#> 2

#> A random question on scheduling with fill in the blank / MCQ.

#> a {suppose it is MCQ, answer is a}

#> Correct. The explanation is below:

- - - -
- -
- - -

#> To attempt another question press 'n'; press 'q' to quit;
press 'r' to return to main menu

#> r

- #> Welcome <user-id> to Online Quiz on OS.

  Select mode:
  - press I for individual mode
  - press G for group mode
  - press A for admin mode

- #> G
- #> Active users are $userId_1$, $userId_2$ --- Specify userId with whom you want to collaborate
- #> $userId_2$
- #> Collaboration established. To send msgs to $userId_2$, @$userId_2$: msg

$userId_2$

#> Request to collaborate from $userId_1$. Press 'y' to affirm, 'N' to reject

#> y

#> collaboration established. Please wait for quiz to begin. To send msg, @$userId_1$: msg

#> Ok. Pick a topic from the following:

1 Threads
2 Scheduling
3 Memory management

} populated in admin mode

#> 2

{ On both user screens }

The same random question on scheduling with fill in the blank / MCQ.

#> @$userId_2$: I think answer is a.

{ suppose it is MCQ }

#> from @$userId_2$: No it should be b

#> b

{ appears on both screens }

#> Wrong. The explanation is below:

— — —
— — — —

#> from @$userId_1$: I think answer is a

#> @$userId_1$: No it should be b ---

. . . .

Similarly, design interface for admin mode to add questions/expln in bulk mode. - Question type, Question text, answer, expln

–

Solution methodology:
You will use Amazon EC2 free tier machine to host the server. You should test your solution with your team mates. For simplicity, collaboration needs to be established only between 2 students.

Assume that each client has a user id whose length is less than 12 bytes. The clients can send different types of messages to the Quiz server. Each message has a fixed format:
16byte header, that indicates the type of message (first 12 bytes) and the length of payload(last 4 bytes); actual message content (payload).

The quiz server reads the header of the message to infer the type of the message and takes appropriate action. Few message type headers are shown below:
1) **myId**- e.g. myId:sriramk this message is sent by the client to the quiz server when it connects to it for the first time. The header is in bold, i.e. myId, and the payload is sriramk.
2) **userId** – e.g. munees:I think answer is a. This will send the message to user id munees.
3) **response –** e.g. response:2, the user has entered 2.

Define the following methods for the client:
-- **initialConnect**(): connects to the quiz server and sends its user-id

-- **getreq**(): reads one line from the user (already defined)

-- **makeHeader** (typeOfMsg, lenOfPayload): creates a 16 byte fixed size header: 12bytes for type of message and 4 bytes for the length of payload e.g. makeHeader("myId", 7): myIdxxxxxxxx padding to make it 12bytes for the type of message and 4bytes to represent length, i.e. 7

-- **readAndSendData**()
  --> **getreq**()
  --> **makeHeader**(...)
  --> send the header followed by payload to the relay server

-- **displayMessage**(): receive messages from relay server and display on the console

Make 2 threads in the client, one for reading input from the user and sending it to server (readAndSendData()), and another one for displaying messages received from the server onto the console (displayMessage()).

Thread1:

```
while (user has not exited) {
        readAndSendData()
}
```

Thread2:

```
while (user has not exited) {
        displayMessages()
}
```

The Relay Server program should also be a concurrent program. It stores the userId and its corresponding socket info. It uses that to transfer messages from 1 client to another.

Each thread of the server program looks like the following:
while (socket not closed)
 -- len,typeOfMsg = readHeader()
 -- msg = readMsg(len)
 -- doAction(typeOfMsg, msg)

The basic code snippet implementing echo server is provided.

You will run the server on Amazon EC2 free tier.

Submission Guidelines:
1) Each group should make 1 submission only. Any member of the group can do that

2) The src file should be named as: RollNo1_RollNo2_RollNo3_RollNo4_CN1.zip

3) Please do not copy. If you have any difficulty, please write to us. We will try to help you out. But if you are caught copying, you will get an F grade in CDP.