OS Final Homework/Project
Fancy Quiz Server

This homework takes the quiz server and enhances it. Some of these enhancements require changes to the server's current behavior, while some are extensions, so please pay close attention to the description, and please ask if anything is unclear or unspecified. This project consists of 3 submissions, one per week.

The new fancy server can manage multiple on-going groups taking different quizzes. In this version, the group admin/creator starts a group and uploads the quiz questions, so the creator is never one of the competitors in its own quiz group.

Whenever a new client accesses the server, the server sends a message about the groups currently awaiting members: OPENGROUPS, followed by the list of groups, each consisting of the topic, the group name, the desired group size, and the current size (number of people who have joined so far). This message is terminated by a CRLF. At any time a client is not involved in taking a quiz, the client may request the current list of open groups, by sending the message GETOPENGROUPS. Also, the server may update a client with any changes to the status of the open groups, providing that the client is not currently taking a quiz, using a different message. The UPDGROUP message tells clients about a new or updated group. The server uses it to let clients (who are not currently taking a quiz) know when a new group is added or a group's current size changes. ENDGROUP is used when a group has ended.

Server sends:	**Client answers:**	**Notes**
`OPENGROUPS|topic|name|size|curr …`	*nothing*	The group information repeats for all open groups
`WAIT`	*nothing*	
`ENDGROUP|groupname`	*nothing*	*Group is ended naturally or unnaturally, but waiting clients are not dropped*
`UPDGROUP|topic|name|size|curr`	*nothing*	

The client may start a new group, or join an existing group, one or the other, not both. To create a group, the client sends the GROUP command, which contains the quiz topic, the group name, and the user's name. The server will respond with SENDQUIZ, which invites the client to send the quiz file as a stream of questions in exactly the same format as in the previous homework. The client QUIZ message is the type of message where the size precedes the data, and the CRLF terminator is not used. This is so the server can read the data in a loop. The server may answer BAD if the group name is already in use (or any other error occurs).

The client may join an existing group with the JOIN message by supplying the group name and the user name. The server responds OK, FULL (group is full), or NOGROUP (group does not exist).

The client may LEAVE a group before or during a quiz. After leaving their current group cleanly, the client may join another group, or create one.

A group creator may cancel a group, only if the quiz has not started yet. If the quiz has started, it will continue, and the creator can exit the program and it will still continue, but the creator cannot cancel it. The LEAVE command is not allowed for the group creator.

Client sends:	**Server answers**	**Notes**
`GROUP|topic|groupname|groupsize`	QUIZ, NAME or BAD	After server responds with QUIZ, the client can send the quiz message
`QUIZ|quizsize|quiztext`	OK or BAD	This message is not terminated with CRLF – the size gives the needed information
`CANCEL|groupname`	OK or BAD	Server closes the group
`JOIN|groupname|username`	OK or FULL or NOGROUP	
`LEAVE`	OK or BAD	
`GETOPENGROUPS`	OPENGROUPS	Server responds with the entire list of open groups

The quiz scoring is the same. For each question, the first person to answer the question correctly is recorded as the winner, and gets 2 points. Other users who get the correct answer get 1 point, no answer gets 0 points, and the wrong answer yields -1 points. Additionally, a user must answer within 1 minute, or be kicked out of the group (same functionality as LEAVE is carried out by force (the socket is not closed).

The group creator is sent each question, but is not allowed to answer. The creator is also sent the message about each question's winner, and about the results. When a quiz is completed, the clients return to being able to create a group or join a group.

The protocol for sending questions and announcing a winner is unchanged. As long as at least one client remains in the group, keep asking questions, until all questions are exhausted. When they are all finished, announce the winner (and standings of all players).

The server sends the size of the entire question plus answers string, and then the string. A terminating CRLF is not used or required because the size precedes the message, and the message contains newlines within it. If the client does not know the answer it sends the special text "NOANS" in place of the answer ID.

After all the users have answered, the server sends the client the winner's name (the name of the client who scored 2 for the fastest right answer). If no one answered correctly, the winner's name sent will be the empty string "".

Server sends:	**Client answers:**
`QUES|size|full-question-text`	ANS|*answerID*
`WIN|name`	

After all the questions are finished, the server sends the quiz standings as a series of names and scores in descending order by score, terminated by CRLF.

Server sends:	**Client answers:**
`RESULT|name|score|name|score …`	*Nothing*

The quiz file format is the same as before. A question may consist of multiple lines, up to a total of 2048 characters. It is separated from the correct answer by a blank line, and the next question follows the correct answer with another blank line in between. Allow for a maximum of 128 questions.

Your server <u>must</u> employ some combination of multiplexing and threaded behavior, which you should choose yourself. For the multiplexing you may use `select` (as we covered in class) or `poll`, but no other technique, for consistency. Your main thread must never block, and should not perform any long-lasting operations. No client should ever be rejected.

Submission Process and Grading

Saturday, April 7 at 23:55

You must submit a neatly formatted document (Word or pdf) that explains your server's architecture and design. It should be at least one page that explains your key data items and data structures, and how your server is going manage them to deliver the quizzes to the groups, and give good, fair service to the clients. Focus on technical details. This should help you plan your own work and serve as a blueprint for your program. This doesn't mean you shouldn't program during the first week.

Saturday, April 14 at 23:55

Submit a draft version of your server. This must be a compilable, runnable version of the server, but it can have bugs, some missing functionality, etc. You must submit a report with it, explaining what is working, what is not, what is still to be done. Also explain what deviations, if any, you have made from your plan, and why.

Saturday, April 21 at 23:55

Submit the final version of the server. Again submit a report explaining what works, what doesn't, and any deviations from the original plan, with explanations.

This project, which will be worth at least 2 homework scores, will be graded not only on correctness and meeting required functionality. A large component of the grade will be the quality of the plan, the consistency of the plan, and your understanding of how and why (or why not) you were able to follow the plan and complete the functionality. So a perfectly functional working server will not necessarily receive 100%, whereas a server with a bug or 2 or a missing feature may get full credit. Also I may call selected students for live grading, and any lack of understanding of the submitted code will naturally lower the grade.

As always, any electronic copying from one another will result in a zero on the assignment (and an F in the class if this is the second time). Any copying from other sources must be clearly and completely cited: it must be clear exactly what section of code came from exactly where. Violations of this rule will result in the same consequences as for copying from another person.