Fancy Quiz Server
Part II. First Draft

Explain 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.

As I intended in my first Architecture and Design mixing of threads and multiplexing was decided be used in this project. Multiplexing via select() system call was used firstly for accepting new clients and I took the example code of echoserver as a starting point and added more features. Furthermore, instead of creating just one thread per group I created two threads:

a) for receiving the quiz from the group creator

```
status1 = pthread_create(&thread, NULL, getQuiz, (void *)fd);
```

b) for starting the quiz only when the needed number of clients arrive:

```
if(groups[leadsock]-> currSize == groups[leadsock]->size){
 status2 = pthread_create(&thread, NULL, startQuiz, (void *)leadsock);
}
```

This second is created in the body of if where JOIN method is handled since each time new client arrives and joins the group, the current size of that group increments.
Also when new group is created leader's socket is removed from the global file descriptors set and added later to the local file descriptors set in each group where quiz is going. The same is with the sockets of the clients who joined groups.

```
FD_CLR( fd, &afds );
// lower the max socket number if needed
if ( nfds == fd+1 )
    nfds--;
 }
```

Let me explain how my data structures changed.

Data structures:

```
typedef struct{
    int mysock;
    int score;
    char name[30];
    char groupName[30];
    int isLeader;
}client;

typedef struct{
    char topic[30];
    int size;
    int currSize;
    char groupName[30];
```

```
    client *members[1024];
    char *lines[1010];
    int numberOfQuestions;
}group;
```

My main data structures did not change much I just added more fields, namely

- `int mysock` to client structure in order for the leader to track the socket numbers of its members;
- `char *lines[1010]` stores the lines of quiz written from the temporary file where server stored the quiz received from the leader.
- `int numberOfQuestions` stores

To store all groups and clients pointer to arrays were created:

```
client *clients[1010];
group *groups[32];
```

Each time when a new group or client is added the memory is dynamically allocated for each element using socket number, which is a file descriptor as an index:

```
clients[fd] = (client*)malloc(sizeof(client));
groups[fd] = (group*)malloc(sizeof(group));
```

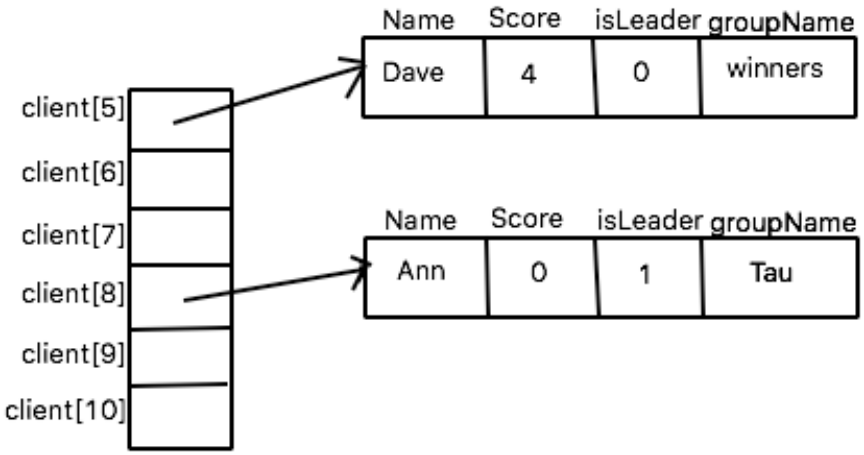

Figure 1. Array diagram for clients

As it is illustrated in Figure 1 each element in clients[ ] points to one client which has attributes: Name, Score, Leader flag and the name of the group to which the client belongs.

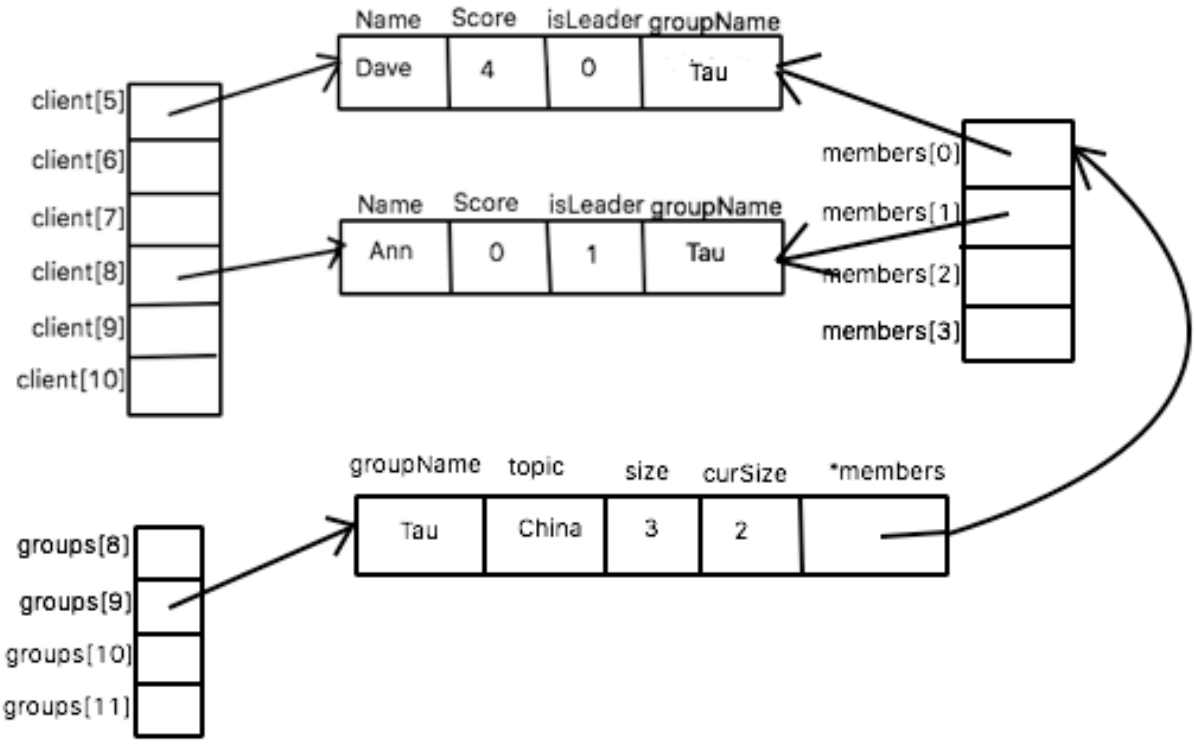

Figure 2. Array diagram for groups/clients

The similar diagram is shown in Figure 2 for groups[ ] array but the last attribute of the groups has client type and it is an array of pointers to the same memory location as from clients[ ] array.

These array diagrams did not change in overall just new fields are added.

Commands from client:

When the client connects to the server the new element in clients[ ] is allocated. Then the server is expecting the answer from the client. Handling of different commands was planned done within if-else blocks, comparing the first token in the client answer's string to existing commands. The list of implemented commands:

1) GROUP command.

Firstly the third token in the string will be checked to ensure that the groupname is not already in use. I am planning here to call the function, which will iterate through the array of groups and return 1 if the groupname is used by other group. If it not there, the client (leader) creates a new element in groups[ ] array, adds topic, groupname and groupsize. Current groupsize is set to 1. Thread is started to receive the quiz from the leader.

2) JOIN command

Client provides name, groupname to which it wants to join and score attribute is set to zero. At the same time a new client should be pointed by the element in members[ ] in that group. To find the group I am planning plan to call here function findByGroupname(), which will be searching from the groups[ ] the group with that name and return -1 if no matching group is found. NOGROUP message is sent. Otherwise, it will return the socket

number of the group leader. Before updating the current size and adding groupname attribute to the client. If the currSize == size then no other clients can be accepted and BAD message is sent.

```
    int findByGroupname(char *grname){
    printf("Join??");
    fflush( stdout );
     for(int i = 0; i < 32; i++){
         if(groups[i] == NULL){ //skip not used socket
             continue;
         }
         if(strcmp(groups[i]->groupName, grname) == 0){
             printf("Found!\n");
             fflush( stdout );
             return i;
     }
}
     return -1;
}
```

As it was mentioned earlier when current size of the group the client just joined achieves group's maximum size, new thread is started to play a quiz in that group.

3) GETOPENGROUPS command

The list of all open group is send iterating through groups[ ] array.

4) LEAVE command

Groupname attribute of the client will be cleaned and the member[ ] is also will be cleaned in groups[ ]. Current size will be decremented. To find the member findByGroupname is again called.
This command is not yet implemented, since I am still thinking how to make the server be ready to accept this command from client at any time.

5) CANCEL command

If the quiz has not yet started group[fd] will be cleaned where fd is a socket of the leader. Server responses with OK only when it is the group leader sending this command and if the currentSize of the group is not equal to the size of the group.

6) QUIZ command

Each quiz is handled inside the thread. When the message SENDQUIZ is sent each thread expects that the client will start to send the quiz probably part by part. Therefore client's response will be read in the loop until the quiz size will be reached. Before the text of the quiz its size sent in the message QUIZ|size|text…..

I will have a variable remSize, which is the quiz's size minus the length of the part of the quiz text sent by the client for the first time.

```
int remSize;

        if(strcmp(splitted[2], NULL) == 0){
            remSize = quizSize - strlen(splitted[2]);
```

```
            printf("%s\n", splitted[2]);
            fwrite(splitted[2] , 1 , strlen(splitted[2]) , fp );
        }else{
            remSize = quizSize;
        }
```

So while remSize is greater than zero server will read from client and each time write the client's response to the temporary file.

Afterwards as it was done in the previous homework by fgets the content of the temporary file was written into array of lines, which is one of the fields of the leader.

It should be mentioned that this code for writing to temporary file will work correctly only for one group since there is only one file called "myquiz.txt" is created where all threads will write their quizzes. In the future I am planning to create a separate temporary file for each thread concatenating their leaders socket numbers with the file name.

Sending the quiz

Concurrency inside the quiz was provided this time by using multiplexing, i.e. the new set of descriptors to hold the participating clients was created and the timer value was set to 1 minute.

Questions are sent one by one. However, here I faced with the problem that each time when one of the clients answers server immediately proceeds to the next question without waiting other clients' responses. This issue should be solved such that the behaviour of the server should be the same as with using conditional variable for threads. Calculating the results and sending to all members is not yet implemented due to above mentioned problem with the synchronized quiz.

Messages send by the server:

7) ENDGROUP message

This message is sent by the server to the clients in the group if leader sends CANCEL message or when the quiz has ended. Group[ ] item is cleaned as well as groupname attribute for all clients who were in this group.

8) OPENGROUPS message

This command is just reply for GETOPENGROUPS command and is sent immediately when the new client is created.

Last note: a numerous amount of error handling should be added after the main functionality will work. Also mutex locks should be used when accessing to global data structures from different threads.