CNT4007C Network Fundamentals
Project 2 – Multi-Threaded Server

Ensure you have a CISE departmental account. Go to https://www.cise.ufl.edu/resources-help/ and IT Help for access to CISE computer systems. If you don't have remote SSH login, consider using PUTTY at https://www.putty.org/

Language: C/C++, Java, Python (however, note that the code must run as specified below.)
Server code must compile and run on storm.cise.ufl.edu; client code must compile and run on storm and on thunder.

Overview

You will extend the basic client-server quiz application using sockets programming to allow for multiple simultaneous clients using a multi-threaded server and timeouts. The server will support creation and administration of contests using questions from a question bank.

Behavioral Requirements

As before, the server will support storage and retrieval of multiple choice questions to and from the main question bank. Different from before, the user will propose a number for each new question, and also for each new contest. This questionID and contestID will be used to reference questions and contests.

There will be two types of clients: contestmeister and contestant. The contestmeister will have a command-line interface that supports handling of contests by creation, adding questions to a contest, and running a contest on the server. The contestmeister will extend the previous client manipulating questions as before (with a few changes), and adding the ability to read commands from a file, create/delete a contest, add questions to a contest, review a contest, and start a contest.

The second type of client is a contestant, and any number of contestants will be able to join a contest that has been started on the server at the behest of the contestmeister. The contestant client will allow the user to supply a nickname for the contestant's user, obtain and print each question as it is given, forward the answer indicated by the user to the server, then get and print the results from the server before getting the next question, until the contest ends.

The client and server will use TCP/IP for communication. The contestant connection will remain open for the duration of the contest that the contestant has joined. The contestmeister connection will remain open until the contestmeister quits or kills the server.

Starting the contest server:

The server will be started from the command line by issuing the command "./cserver" and will print a random port number (<CMport>) it obtains from the system to stdout. The port number printed to stdout will be the port for the contestmeister only (for Java and C++, using port 0 allows the system to make the port selection, but python does not work this way. Under no circumstances hardcode the port.) The port number along with the hostname will enable the contestmeister to connect to the server. If the server fails to open a port it will print an error message to stderr and terminate. Run the server in the foreground (i.e., do NOT use the '&' to run it in the background, so you will need a Putty window per process) so that when the terminal closes, the server will also shut down (avoiding headaches with excess processes).

The contest server will accept control messages from the contestmeister client (manipulating questions and contests), and contest activity from the contestant clients. See the contestmeister commands for information on what it must do in response to contestmeister commands. The contestant client will not issue any commands to the contest server, but will only supply a nickname, then receive questions and results, and supply answers until the contest is over.

Shutting the server down:
The server can be shut down as before by accepting a shutdown command from the contestmeister client. In this case the server will print a shutdown message to stdout and terminate. Since the server will be run in the foreground, you may also shut it down using control-D or via job control.

Starting the contestmeister client:
The contestmeister client will be started from the command line by issuing the command "./contestmeister <host> <CMport> [<cmdfilename>]" where <host> is the hostname on which the server is running and <CMport> is the port that the currently running server obtained from the system and printed to stdout. The optional command line argument <cmdfilename>, if present, will name a file that contains commands in the format used for the contestmeister to add questions to the question bank and to create contests. These commands will be run after the contestmeister connects to the cserver, and before accepting any commands from the user.

Starting the contestant client:
The contestant client will be started from the command line by issuing the command "./contestant <host> <Cport>" where <host> is the hostname on which the server is running and <Cport> is the port that the contestmeister obtained from the currently running server and printed to stdout (see below). The contestant client shall not connect to the contestmeister port, which remains open on the server, but only on the port that is associated with a particular contest.

If either client fails to make a connection with the server running on that host at the designated port, it shall print an error message to stderr and terminate.

Contestmeister Client Commands

The contestmeister program will accept the following commands on a command line. The client will print the prompt "> ". (The prompt is similar to a Unix prompt. With a Unix prompt, if you hit a carriage return(enter), you just get another prompt. Don't allow the program to crash if there is a carriage return and no input provided. That will result in a deduction.)

Commands from first project:

p <question #>: put a question in the bank with number <question #> – p is followed by the proposed question number and then the question fields as shown below (this is different from project 1). When finished, display the question number accepted by the server for the question or an error message. (Note: this has changed for the user to supply the question number – if the question number is already in use, the server shall reject it and the client shall print an error message "Error: question number <question #> already used." On success, the client shall print a success message "Question <question #> added.")

Question fields (each on its own line as before):
question tag – string that may have spaces, commas, etc. but not a new line or unprintable char
question text – one or more lines of text, followed by a line containing a period '.' by itself
question choices – two or more choices, with the last choice followed by a lone period line
choice – each choice is one or more lines of text followed by a lone period line
each choice text shall start with "(<x>)" where <x> is a consecutive letter a, b, …
correct answer - <x> where <x> is the letter of the correct answer.

Example (comments are shown starting with // and are not part of the actual example)
Output from the contestmeister is indicated in bold, but this is hard to see for the prompt.

```
> p 23                                   // user now supplies the question number
presidents, US history                   // this is the question tag
Who was the first president of the USA   // this is the question text
.                                        // the period is user supplied manually typing it in
(a) Thomas Jefferson                     // first question choice,
                                         // the (a) supplied by user
.                                        // this ends the first choice
(b) Abraham Lincoln                      // second choice …
.
(c) George Washington                    // third choice …
.
(d) Benjamin Franklin                    // fourth choice …
.                                        // this period ends the fourth question choice
.                                        // another period ends the list of choices
c                                        // the correct answer choice is (c)
Question 23 added                        // Success message from cserver's acceptance

> p 23                                   // duplicate question number
astrophysics                             // tag
The moon is made of green cheese.        // T/F question
.                                        // period ends question text
(a) True                                 // choice (a)
.                                        // end of choice (a)
(b) False                                // choice (b)
.                                        // end of choice (b)
.                                        // end of all choices
b                                        // correct answer is (b)
Error: question number 23 already used   // Error message from cserver's rejection
>                                        // blank line does not crash client
> p 2                                    // new question number
astrophysics                             // tag – same question
The moon is made of green cheese.        // T/F question
.                                        // period ends question text
(a) True                                 // choice (a)
.                                        // end of choice (a)
(b) False                                // choice (b)
.                                        // end of choice (b)
.                                        // end of all choices
b                                        // correct answer is (b)
```

```
        Question 2 added                          // Success
>
```

d <n>: delete a question from the bank – delete question number <n> from the bank. Display either an indication of success or failure.

Example:

```
> d 2                                       // delete question number 2
Deleted question 2                          // indication of success from cserver
>
```

g <n>: get question number <n> from the bank. Display the question in the format given for the p command, or an error message if there is no question with that number.

Example:

```
> g 2                                       // get question number 2
Error: question 2 not found                 // error message: cserver rejection
> g 23                                      // ask cserver to get question 23
presidents, US history                      // this is the question tag
Who was the first president of the USA?     // this is the question text
.
(a) Thomas Jefferson                        // this is the first question choice
.                                           // this ends the first choice
(b) Abraham Lincoln
.
(c) George Washington
.
(d) Benjamin Franklin
.                                           // this ends the fourth question choice
.                                           // this ends the list of choices
c                                           // the correct answer choice is (c)
>
```

r – this command is repurposed in contestmeister

c – this command has been eliminated from the client

k: kill – terminate the server and then terminate the client.

q: quit – terminate the client. The server must gracefully drop the socket and get ready for a new connection

h: help – print brief instructions on these commands

New commands for the contestmeister:

s <contest #>: set <contest #> – set a new contest. Create a new contest with number <contest #>. If <contest #> is already in use, print an error message to stderr, otherwise print an acceptance message.

Example:

```
> s 5
Contest 5 is set
> s 5
Error: Contest 5 already exists
> s 6
Contest 6 is set
>
```

a <contest #> <question #>: add <contest #> <question #> – append question <question #> to contest <contest #>. If either number is invalid, print an error message to stdout, otherwise, append the question to the contest. The server has the master question bank; this command draws from the bank to add the question <question #> to contest <contest #>. The same question may be added to more than one contest.

Example:

```
> a 3 23
Error: Contest 3 does not exist
> a 5 23
Added question 23 to contest 5
> a 5 22
Error: Question 22 does not exist
>
```

b <contest #>: begin <contest #> - begin a new contest. If the contest number is invalid, print an error message to stderr. Otherwise, the cserver will obtain a new, random port number (<Cport>) from the system and print it to stdout with contest number (below the CM port). The cserver will also reply to contestmeister (CM) with the port number, which the CM will also print to stdout with the message "Contest <contest #> opened on port <Cport>, where <Cport> is the port on the cserver to which the contestants will connect. The user running the contestmeister must communicate the cserver hostname and the contest port number to any users who will participate in the contest.

The cserver will then start accepting contestant connections on that port to participate in contest <contest #> for a period of one minute. Once the one-minute timer has expired, the server will not accept any more contestants, and will start the contest.

After each contestant has connected, the cserver will accept from the contestant client a nickname for that client, ensure it is unique for that contest, and return a reply to the client indicating that the nickname has been accepted or rejected. Once a client has a valid nickname, the client waits for questions.

Once all the clients have connected and obtained a unique nickname for that contest, the cserver will start to give the clients the questions in the contest. For each question in the contest, in order, the cserver will send the question (without the answer) to each contestant, then wait for an answer from each contestant. When all the contestants have provided an answer, the cserver will send to each contestant their result (correct or incorrect) and percent correct answers from all the contestants for that question. It will also send each contestant its current score (# correct answers) and the current maximum score. Finally, it will send each contestant the next question (if there is one), or send an indication that the contest is over.

When the results of the last question have been sent to each client, the cserver will close the port associated with the contest, as will each contestant client. After printing the final results, the contestant client will terminate.

l: list – list contests by contest number. Output the numbers of existing contests, one per line, followed by a tab, then the number of questions, whether the contest has been run, and if so, the contest statistics: average and maximum correct answers in the most recent run of the contest.

Example:

```
> l
5	1 question, run, average correct: 0.5; maximum correct: 1
6	0, not run
>
```

r <contest #>: review <contest #> - review a specific contest. If a contest with that number exists, then output the contest number followed by a tab, then the number of questions in the contest, whether the contest has been run, and if so, the overall statistics (average and maximum correct). Then for each question in the contest, in order, on a separate line, print a tab, then the questions number, and if the contest has been run, another tab followed by the percent correct answers for that question in the most recent run of the contest.

Example:

```
> r 3
Error: Contest 3 does not exist
> r 6
6	0 questions, not run
>
> r 5
5	1 question, run, average correct: 0.5; maximum correct: 1
	23	50% correct
>
```

Contestant Client:

The contestant client is started by typing on the command line "./contestant <host> <Cport>" where <host> is the hostname on which the cserver is running and <Cport> is the contest port number that the currently running server obtained from the system and provided to the contestmeister when the contest was started when the contestmeister executed the 'b' command, and which the contestmeister printed to stdout for the user running the contest to give to the users participating in the contest using the contestant clients. If the contestant client is unable to connect to the cserver on the named host and port, then the contestant client will print an error message to stdout and terminate.

There are no commands that the contestant user needs to issue. However, the contestant client program will prompt the user for the desired input and print the output from the cserver. The first prompt requests the nickname of the user for that contest. This is sent to the cserver, and if it is not already taken, the cserver accepts the name. Otherwise, the cserver rejects the nickname and the client must print out the rejection and prompt for a different nickname.

Once the nickname has been accepted, the contestant client shall wait for the cserver to send it the first question. The client shall print the question out in suitable form (see below) and prompt the user to select an answer. When the user selects a valid answer, the client will send it to the cserver. When all the contestants have sent their answers to the cserver, the cserver shall send each client the result (correct or incorrect), the percent correct answers from all the contestants for that question, the particular contestant's current score (# correct answers so far out of #questions so far) and the current maximum score over all contestants. The client will print this information to stdout and wait for the next question. This cycle will continue until the contest is over, as indicated by the cserver to the client when it is waiting for the next question.

Example Run:
(the system prompt is indicated by $, comments by //)
(System and process output is indicated by boldface type.)

[Cserver run on storm terminal 1:]

```
$ ./cserver                          // see below for contents of contest1.txt
5678                                 // port number obtained by server for contestmeister
Contest 1 started on port 6789       // printed after the CM runs the first b command (see below)
Contest 1 started on port 7890       // printed after the CM runs the second b command (see below)
…
```

[Contestmeister on thunder terminal 1:]

```
$ ./contestmeister 5000
Error: invalid port number: 5000  // error message for bad port number
$ ./contestmeister 5678 contest1.tx  // right port number as printed out by cserver
                                     // along with file containing commands to build contest1
> l                                  // list contests
1       2 questions, not run         // contest 1 has 2 questions, has not yet been run
> b 1                                // begin contest number 1, loaded by contest1.txt
6789                                 // contest number 1 is assigned port 6789
                                     //user running CM knows port 6789
                                     //has to tell users who will run the contestants
> b 1                                // start another instance of contest 1
7890                                 // second instance of contest number 1 is assigned port 7890
…
```

[Contestant #1 run on storm terminal 2:]

```
$ ./contestant storm 6789                  //user got info from CM, client can run anywhere
```

```
Please input a nickname: Moe<cr>          // prompt from client,user enters Moe
                                          // "Moe" sent to Server, which checks nickname to
                                          // see if name already in use
Hello Moe, get ready for contest!         // nickname accepted
Question 1:                               // Display first question received from cserver
The moon is made of green cheese.
(a) True
(b) False
Enter your choice: a                      // Prompt user for choice, send to cserver
Incorrect. 50% of contestants answered this question correctly.      // print results
Your score is 0/1. The top score is currently 1/1.

Question 2:                               // Display next question received from cserver
Who was the first president of the USA?
        (a) Thomas Jefferson
        (b) Abraham Lincoln
        (c) George Washington
        (d) Benjamin Franklin
Enter your choice: a
Incorrect. 50% of contestants answered this question correctly.
Your score is 0/2. The top score is currently 2/2.
The contest is over – thanks for playing Moe!     // Contest is over
$
```

[Contestant #2 run on thunder terminal 2, slightly after the first bit of contestant dialog on terminal 1:]

```
$ ./contestant storm 6789
Please input a nickname: Moe<cr>          // prompt from client,user enters Moe
                                          // "Moe" sent to Server, which checks nickname to
                                          // see if name already in use
Error: Nickname Moe is already in use.    // Error message from rejection by Server
Please input a nickname: Steve<cr>                // prompt from client, user enters Steve
Hello Steve, get ready for contest!       // Nickname is accepted
Question 1:                               // Same kind of dialog as with the first contestant, Moe
The moon is made of green cheese.
(a) True
(b) False
Enter your choice: b
Correct. 50% of contestants answered this question correctly.
Your score is 1/1. The top score is currently 1/1.

Question 2:
Who was the first president of the USA?
        (a) Thomas Jefferson
        (b) Abraham Lincoln
        (c) George Washington
        (d) Benjamin Franklin
Enter your choice: c
Correct. 50% of contestants answered this question correctly.
Your score is 2/2. The top score is currently 2/2.
```

The contest is over – thanks for playing Steve!
$

Submissions:
You will submit
a) all source code for cserver, contestmeister, and contestant. This code will compile (as appropriate) and run on storm.cise.ufl.edu and thunder.cise.ufl.edu after make (1) is run.
b) a makefile with targets cserver, contestmeister, and contestant that produces the cserver, contestmeister, and contestant executable files that run the server and the clients from the command line after “make all” is run from the command line on storm.cise.ufl.edu (and on thunder.cise.ufl.edu for contestant). Make contestant shall only make the contestant executable.
c) PDF manual for the system (does not have to be man pages), including cserver, contestmeister, and contestant describing the use of the system as you built it.
d) Any test files that you used to test your system as documented in the report.
e) Report that documents the system (text, pdf, doc, or docx) – include a description of the code, any important structures, classes, variables, methods, functions, etc. and how they are used. Describe the architecture (client-server) and **completely document the protocol** used between clients (both types) and server, including message structure, actions taken before/after each message, and error handling. This means providing message structure and meaning, when the message is sent, by whom and to whom, and actions taken on message receipt. If your server uses a database facility rather than a local file, then document that here also. Include a section on unit testing and acceptance testing that shows all the tests you ran to verify your code.

All files will be uploaded on canvas (using zip, tar, or rar, as appropriate).

Advice and Suggestions:
Be sure your code compiles and runs correctly on storm.cise.ufl.edu and thunder.cise.ufl.edu or you will not receive credit for this assignment.

If you are using an interpreted language that requires the interpreter to be run with the code as its argument, then use the makefile to create script files with the right contents to do this and execute correctly according to the directions above. Remember to make the script executable!

Example load file contest1.txt:
```
p 23
presidents, US history
Who was the first president of the USA?
.
(a) Thomas Jefferson
.
(b) Abraham Lincoln
.
(c) George Washington
.
(d) Benjamin Franklin
.
.
```

c
p 27
astrophysics
The moon is made of green cheese.
.
(a) True
.
(b) False
.
.
b
s 1
a 1 27
a 1 23