CPSC 3500 Computing Systems
Winter 2018

Programming Assignment 4: Client-Server Programming
Due: Monday, March 19 at 2:05pm

Description

In this assignment, you will be creating a client and server program to play a simple number guessing game. Client programs can connect to the server to play the game. Clients send their guess to the server. The server responds by sending the current game status back to the client.

The game requires user to guess three mystery numbers (in order) that range from 1 to 200.

Here is the process to play the game:

1. The player (client) is prompted to enter the name and is sent to the server.
2. The three numbers are selected randomly by the server.
3. The player enters three numbers as a guess.
4. The client makes sure the guess is valid and sends the guess to the server.
5. The server first checks if the player has successfully guessed all three numbers. If so, go to step 8.
6. If the guess was not correct, the server responds to the client with how many of the guesses were too high, how many guesses were too low, and how many guesses were correct.
    - The server does not give information on specific guesses – only the total is given. If the mystery numbers are 90, 123, 39 (in that order) and the player guesses 60, 90, 50 (in that order), the server responds that two guesses are too low (60 < 90, 90 < 123) and one guess is too high (50 > 39).
7. Repeat steps 3-6 until the game has won.
8. When the game is won, the server will respond with a victory message that indicates the number of turns it took to guess the mystery numbers.
9. In addition, the server will send a leader board indicating the name of the top three players along with the number of turns it took to guess the mystery numbers.

Number Guessing Client (`hw4_client.cpp`)

Here is a more detailed description of the client:

1. Start the client which has two *required* command line arguments (in this order):
    - IP address of the server
    - port of the server process (use one of the ports assigned to you)
2. Connect with the server. Exit with an appropriate error message if a connection could not be established.
3. Ask the user for their name.
4. Send the name to the server.

5. Ask the user for a guess. A guess consists of three integers. You may assume the user always enters three integers.
6. Check if the guess is valid (all numbers are within the 1-200 range). If the guess is invalid, print an error message and go back to step 5.
7. Send the guess to the server.
8. Receive the result of the guess from the server.
9. If the guess was unsuccessful, print the result of the guess to the screen and go back to step 5.

Note: The remaining steps are only carried out if the guess was successful.

10. Receive an additional victory message from the server.
11. Print the victory message to the screen.
12. Receive leader board information from the server.
13. Print leader board information to the screen.
14. Close the connection with the server.

Number Guessing Server (`hw4_server.cpp`)

Here is a more detailed description of the multithreaded server:

1. Start the server which has one *required* command line argument:
    - port number (use one of the ports assigned to you)
2. Listen for a client.
3. When a client connects, *create a new thread* to process that client (see below).
4. The server will run forever until aborted (Ctrl-C).

The following steps correspond how to process each client:

1. Generate three random numbers from 1-200.
2. Print the numbers to the screen (while this is not necessary for actual gameplay, it helps debugging and grading).
3. The first communication step is to receive the player's name from the client.
4. Receive a guess of from the client.
5. If the guess is not successful, send the result of the guess and go back to step 4.

Note: The remaining steps are only carried out if the guess was successful.

6. Send the result of the guess (all three numbers are correct).
7. Send a victory message indicating the number of turns it took.
8. Update the leader board if the player made the top three.
9. Send the current leader board to the client.
10. Close the connection with the client.

Leader Board

A few notes about the leader board:

- When the server is started, the leader board initially starts empty.
- If the leader board contains fewer than three entries, only display the current number of entries.
- Ties are broken by whoever completed the game first as determined by when the leader board is updated at the end of the game.
    - For example, let's say the following people played in the following order from earliest to latest with number of turns in parentheses: Anna (19), Maya (15), Mark (19), John (15). Then the leader board would be:

      1. Maya 15
      2. John 15
      3. Anna 19

    - Do not make this more complicated than necessary. This rule actually makes tie handling very simple.
- The output of the client should display each person on the leader board on its own line with each line displaying the user name and the number of turns it took (separated by a space). See the example above.
- Did you remember that the data structure storing the leader board is shared among the different client threads? Your program should protect against any issues that may arise from sharing the leader board.

Compiling and Running your Program

You will need to create a **`Makefile`** that compiles your client and server programs. Since you are creating two programs, it is necessary to use the **`-o`** option in g++ to give your client and server different names: **`pa4_client`** and **`pa4_serve`**r respectively.

To run your server and client, it is necessary to have two terminal windows open. In one window, start the server (the server must be started before the client). Then run the client in a different window. The IP address for `cs1` is `10.124.72.20`. Use the ports assigned to you in class. Here are some sample command lines assuming the client program is called **`pa4_client`**, the server is called **`pa4_server`**, and one of your assigned ports is 10000:

```
./pa4_server 10000
```

```
./pa4_client 10.124.72.20 10000
```

Before submitting your program, you should test your server with two or more clients connected to the server at the same time. Each client will need their own window.

If either program terminates (either Ctrl-C or program error) while the connection is still active, you may get a bind error. If this happens, select a different port. In this situation, the OS thinks the port is still being used and it takes a minute or two for the OS to figure out the process using that port has terminated.

Sample Playing of the Game

For this sample playing, assume that the three mystery numbers are 173, 46, 105.

```
Welcome to Triple Number Guessing Game!
Enter your name:  Eric

Turn: 1
Enter a guess of three numbers: 100 100 100
Too high: 1  Too low: 2  Equal: 0

Turn: 2
Enter a guess of three numbers: 150 46 150
Too high: 1  Too low: 1  Equal: 1

Turn: 3
Enter a guess of three numbers: 180 46 105
Too high: 1  Too low: 0  Equal: 2

Turn: 4
Enter a guess of three numbers: 173 46 105
Too high: 0  Too low: 0  Equal: 3

Congratulations! It took 4 turns to guess all three numbers!

Leader board:
1. Maya 2
2. Eric 4
3. Anna 12
```

Implementation Notes

For the most part, it is up to you to determine how the various information is transmitted over the network. You may assume that the client only connects to a program running your server and vice versa. However, there are a few rules:

- The guess must be sent as three integers in network order.
- The response (result of the guess) must be sent as three integers in network order.
- You may send strings using any method you would like. However, you may NOT place a limit on the number of characters on the user's name.
- The leader board can be sent as a series of records or as a long string (easier).
- The server must generate new random numbers for each client.

Error Checking

- Since the server runs forever, it must be robust. In particular:
    - Any error that occurs when processing a client causes only that thread to abort. These types of errors should not cause the server as a whole to crash.
    - The server should be free of leaks – dynamically allocated resources need to be reclaimed.
- If a socket function that creates / uses the listening socket fails, abort the server with an appropriate error message.

Submitting your Program

On **`cs1`**, run the following script in the directory with your program:

`/home/fac/elarson/submit/cpsc3500/pa4_submit`

This will copy all source code files (must have a **`.cpp`** or **`.h`** suffix) in the current directory to a directory that accessible by the instructor.

In addition it will look for a makefile (with the file name **`Makefile`**). If a makefile is not present, the submission script will reject your program. The submission program will then attempt to compile your program using **`make`** and will look for executable **`pa4_client`** and **`pa4_server`**. If **`make`** fails and/or the required executables are not present, the submission program will reject your submission. If you are unfamiliar with using make and/or creating make files, please consult the tutorial on the class website.

Failure to submit a successful submission before the due date and time will result in a zero. Programs that fail to compile, even if make is successful, will result in a zero. Only the last assignment submitted before the due date and time will be graded. *Late submissions are not accepted and result in a zero.*

Grading

The assignment will be graded with the rubric on the following page. The following additional deductions are possible:

- Failure to use command line parameters (and asking the user for the values instead) will result in a 15 point penalty. If you need help on using command line parameters in C++, please consult the brief tutorial posted on the course website.
- Using the command line but temporarily using a hard-coded IP address or port (and not fixing the issue before submission) will result in a 6 point penalty.
- Programs that do not compile and are rejected by the submission program will receive a zero.

Grading Rubric

Feature	Max Points
Client • Server properly display mystery numbers. (3 points) • Generates new random numbers for each client. (2 points) • Properly returns correct result of each guess. (8 points) • Properly displays winning message with correct number of turns. (7 points) • *Properly displays leaderboard. (7 points) • No other issues associated with client (3 points)	30
Leaderboard • *Proper functionality. (14 points, potential deductions shown below) o Only shows one name (and more than one person has won): -12 o Incorrectly updated (not related to ties): -6 (or more) o Incorrectly handling ties: -3 • *Proper formatting (4 points) • Handle concurrent updates properly. (7 points)	25
Networking • Server is multithreaded and can handle concurrent clients. (8 points) • Guesses and the results of guesses are sent as integers in network order. (4 points) • Name and leaderboard are sent as arbitrary length strings. (8 points)	20
Error checking / handling • Five tests (3 points each)	15
Style • Style will be inspected manually based on the guidelines within the programming assignment expectation handout. Meeting all guidelines will result in full credit. • As noted on the programming assignment expectation handout, debugging code and extraneous output should be removed from the client before submission – failure to do so will result in a style deduction. • However, it is permissible for the server to print an occasional status message (such as when the client connects). In fact, it is a requirement that the server print out the three mystery numbers at the start of each game.	10
TOTAL	**100**

*If leaderboard is not properly displayed on the client at all, the leaderboard is assumed to have improper functionality and incorrect formatting. The assignment would then incur all three deductions totaling 25 points.