CSCI 4211: Introduction to Computer Networks

Fall 2025

PROGRAMMING PROJECT 1: Trivia Game

Due: October 14th, 2025, at 11:59 p.m.

Note: You can use generative AI tools for this project.

1. Overview

In this project, you are tasked with implementing a trivia game application with a server that allows multiple clients to play simultaneously. This project aims to deepen your understanding of protocols, socket programming, JavaScript Object Notation (JSON), and the cloud. Your solution **must run** on a Linux machine.

2. General Client-Server Interaction

You are required to create a client-server application for a trivia game that asks five randomly selected questions from **`trivia_questions.json`** (see Section 5.2) about the University of Minnesota. A general interaction between the client and server should proceed as follows:

1. The server is waiting for a new connection.
2. The client connects to the server to play a game.
3. The server sends the client a randomly selected question and its possible choices.
4. The client replies with the user's answer.
5. The server scores the user's answers and sends the result to the client.
6. Steps 3 - 5 are repeated until the remaining four trivia questions have been asked and answered.
7. The server will send the user's total score to the client and wait to hear from the client whether the user wants to play another game. If they do, then the server reuses the same connection and starts a new game. If not, then the connection is closed.
8. The client displays the user's total score and asks the user if they want to start a new game. If they do, then the client indicates this to the server and keeps the previous connection open to play another game. If not, then the connection is closed.

Notes:

- Your server must be able to handle multiple clients simultaneously. This can be implemented through threads or a similar mechanism. The choice is up to you.
- Specific technical details are intentionally excluded to encourage self-directed exploration through official documentation and external resources.

3. Deploy the Server in the Cloud

You should first run your client and server on the same machine to make testing and debugging easier.

Once your code can handle multiple games being played simultaneously on the same machine, you can move on to deploying your server in a cloud environment so that a remote client with Python 3 installed can play. Use the free trial for a public cloud provider (i.e., Google Cloud, Amazon AWS, or Microsoft Azure) to create a VM and run the server code in a manner that allows it to continue running even after you close out of your SSH session.

Hint: Ensure that you create firewall rules that allow for traffic to and from your server from anywhere in the world. During grading, the instructors will attempt to connect to your cloud server using your client code, so this must be implemented correctly to work.

We recommend that you test your cloud server by running multiple remote clients simultaneously to ensure that your code still works as expected.

For full points (see Section 7.1), your cloud server must be running by the deadline, and your client code must work out of the box without modification from the instructors.

4. Write a Report

Write a report containing the following information:

- Q1: How did you implement the functionality to allow the server to handle multiple clients simultaneously?

- Q2: Did you successfully deploy your server in the cloud? If so, include 1-2 screenshots of the cloud provider dashboard showing your virtual machine fully set up and running. What is the public IP address and port number your server is running on in the cloud?

- Q3: Summarize your experience setting up your virtual machine and running your server in the cloud. Did you face any issues/challenges? If so, then how did you attempt to resolve them, and did your attempts succeed?

5. Provided Starter Files

5.1 Skeleton Code

You are provided with starter code files that are written in Python (**`client.py`** and **`server.py`**). The provided client and server code are skeletons that you must complete according to the instructions in Sections 2 and 3. These files include some NOTE comments to indicate code that shouldn't be modified.

Your client and server code should work when they're executed on the same machine or the server is running in the cloud. The only difference is the server's host and port information that's used by the client and server. This information can be easily switched by providing the correct command-line argument (see Section 6.2).

5.2 Trivia Questions File

As mentioned in Section 2, you are provided with a file (**`trivia_questions.json`**) containing ten trivia questions about the University of Minnesota, where five should be randomly selected for each new game. JSON files have a key-value format. The following is the first key-value pair in the **`trivia_questions.json`** file.

```
"1": [
            "What year was the U of M established?",
        [
            "1803", "1812", "1851", "1923"
        ],
            "3"
]
```

Here is a breakdown of the various parts that you will need to understand:

- **`"1"`** is the key that must be used to access the associated value.
- **`["What year was the U of M established?", ["1803", "1812", "1851", "1923"], "3"]`** is the value.
- **`"What year was the U of M established?"`** is the trivia question.

- `["1803", "1812", "1851", "1923"]` is a list of the possible answers to the trivia question.
- `"3"` is the position of the correct answer in the list of possible answers, which would be `"1851"` in this case.

6. Additional Help

6.1 Helpful Resources

While almost *any* reference can be helpful to you, the following should be particularly useful when completing this project:

- Classes
- Official Socket Documentation
- Socket - Low-level Networking Interface
- Socket Programming in Python (Guide) - Lots of sample code
- Read JSON file using Python - GeeksforGeeks
- JSON encoder and decoder — Python 3.13.2 documentation
- How to Use the nohup Command in Linux | DigitalOcean

6.2 Code Execution Steps

We recommend using VSCode to implement this project because it offers many useful features and simplifies programming. In whatever IDE you implement this project in, you will need to follow these steps in the given order to execute the code:

1. Open two separate terminals.

2. Run the server code in one terminal:
    - Local server: `python3 server.py 1` OR `python server.py 1`
    - Cloud server: `python3 server.py 2` OR `python server.py 2`

3. Run the client code in the other terminal:
    - Local client: `python3 client.py 1` OR `python client.py 1`
    - Remote client: `python3 client.py 2` OR `python client.py 2`

4. In the client terminal, enter the input according to the prompts.

Notes:

- The server must always run before the client(s). Otherwise, the client(s) won't have anything to connect to, which will cause the program to crash.

- If you want to test multiple clients, then just open another terminal and run the same client program (i.e., one terminal for the server and two terminals for two clients).

6.3 Server Process Termination

Typing **`Ctrl+C`** in the server's terminal will trigger a signal handler that will gracefully shut down the server by releasing any allocated resources and closing open files.

If you fail to kill the server process properly using **`Ctrl+C`**, then the port the server was running on will still be in use and can't be immediately reused in your next execution of the server. Therefore, another way to kill the previous server instance is by following these steps:

1. Find the process ID (PID) of the server: **`ps aux | grep python`**

2. Terminate the server process: **`kill -9 <PID>`**

6.4 Terminal Output

6.4.1 Client

The skeleton client code includes printing of the application banner, the game instructions, the "Game Over" header, and handling of the user's initial inputs. This provided code shouldn't be modified. However, you are responsible for formatting the trivia questions, their possible answers, and the scoring results received from the server. Below are examples of how you could display this information, but you're not required to copy them. We'll accept anything as long as it's readable and easy to follow.

```
Question 1: How large is the U of M Twin Cities campus (in acres)?
----------------------------------------------------------------------
  1) 2,730
  2) 3,610
  3) 5,450
  4) 7,080

Answer: 1
Correct answer! You scored 1 point.
```

```
Question 2: Who is the current U of M president?
------------------------------------------------------------------
  1) Joan T.A. Gabel
  2) Rebecca Cunningham
  3) Robert H. Bruininks
  4) Eric Kaler
  5) William Watts Folwell

Answer: 1
Wrong answer. The correct answer was 'Rebecca Cunningham'.
```

```
---------------------------- GAME OVER -----------------------------
Total score: 1 point

Thanks for playing!

Would you like to start a new game? [y/n]:
Input: n

Client is exiting...
```

6.4.2 Server

The server code in your final submission shouldn't have much of any terminal output beyond statements about server initialization and shutdown. Your code can have debug statements printing out a unique identifier for the client, followed by what the server received and responded with, or the scoring results of the user's answer (see image below for examples). However, all debug statements should be deleted or commented out in your final submission to not clutter the autograder's output.

```
Server is listening...
(('127.0.0.1', 35880)) Starting a new trivia game...
(('127.0.0.1', 35880)) Correct answer! You scored 1 point.
(('127.0.0.1', 35880)) Wrong answer. The correct answer was '19'.
(('127.0.0.1', 35880)) Wrong answer. The correct answer was '7,100'.
(('127.0.0.1', 35880)) Wrong answer. The correct answer was '1851'.
(('127.0.0.1', 35880)) Wrong answer. The correct answer was 'Rebecca Cunningham'.
(('127.0.0.1', 37550)) Starting a new trivia game...
(('127.0.0.1', 37550)) Correct answer! You scored 1 point.
(('127.0.0.1', 37550)) Correct answer! You scored 1 point.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was 'St. Cloud'.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was '12'.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was '1935'.
(('127.0.0.1', 37550)) Starting a new trivia game...
(('127.0.0.1', 37550)) Wrong answer. The correct answer was '12'.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was 'Rebecca Cunningham'.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was '26th'.
(('127.0.0.1', 37550)) Correct answer! You scored 1 point.
(('127.0.0.1', 37550)) Wrong answer. The correct answer was '1851'.
^C
Received signal:  2
Performing cleanup...
Exiting gracefully.
```

6.5 General Advice

Get Started Early! Don't wait until the last minute. Starting early will allow you the opportunity to have more time to promptly receive any help that you may require from the instructors and to calmly debug your code if any unexpected and/or difficult problems arise.

7. Submission Information

7.1 Grade Breakdown

- (+ 10) Code Documentation and Readability
- (+ 10) Code Correctness and Structure
- (+ 10) Report
- Test Cases
    - (+ 20) One client; client and server are running on the same machine
    - (+ 10) Multiple clients; clients and server are running on the same machine
    - (+ 20) Setting up cloud server
    - (+ 10) One client; client is local, and the server is running in the cloud
    - (+ 10) Multiple local clients; clients are local, and the server is running in the cloud

7.2 Coding Expectations and Guidelines

Before submitting, check your work against the "Coding Expectation and Guidelines" document linked under the Important Course Links module on Canvas. Your code won't be strictly graded based on these expectations and guidelines, but you should still generally follow them as they include best practices. However, be aware that extremely poor code implementations *will* negatively affect your grade since this is an upper-division course.

7.3 Gradescope Autograder Details

There are five test cases in the Gradescope autograder:

- The first three test cases simply check for the existence of the three required files in your submission (see Section 7.4).
- The fourth test case runs your server code and one instance of your client code on the same machine.

- The fifth test case runs your server code and multiple instances of your client code on the same machine.

Points aren't assigned to these test cases because the output needs to be manually inspected and graded by the instructors. You should still ensure that your submission passes all of the test cases before the deadline. If the test case fails and it's before the deadline, then you can take the opportunity to try to fix the problem(s) and resubmit as many times as you want until you pass all of them.

There are no test cases for the cloud deployment of your server, as this will be manually tested by the instructors using your client code.

7.4 What To Submit

You must submit to the class Gradescope website **a ZIP file** (.zip format), named <StudentID-FirstName_LastName>-project1.zip (example: 1234567-Alice_Jones-project1.zip), that contains the following files:

1. Your modified client source code file (as **`client.py`**), which **MUST** run on a Linux machine.

2. Your modified server source code file (as **`server.py`**), which **MUST** run on a Linux machine.

3. A PDF report (as **`report.pdf`**). Refer to Section 4 to see what this report should contain.

Note: Do not include any subfolders in your submission. Manually select all of the required files and then ZIP them rather than zipping the folder that contains them. If you ZIP your files incorrectly, then the autograder test cases **will** fail.