CSCI 136 Assignment – Shared, Networked Canvas

March 29th, 2023

Due Date: Thursday, 4/6/2023 11:59PM

Points: 50

Topic: Networking

In this assignment, you will create a client and server program that allows multiple clients to draw on a shared canvas. In the process you will develop network client and server programs using socket communication. This includes creating a multi-threaded server that can handle concurrent requests from multiple clients.

After following all instructions below, please submit your Python scripts (`SharedCanvasServer.py`, `SharedCanvasClient.py`) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, and other useful information. Make sure classes, methods and functions are thoroughly commented.

- `SharedCanvasServer.py`
- `SharedCanvasClient.py`

Grading:

This assignment is worth 50 points. You will be graded according to the following criteria:

Grade Item	Points
Scripts Run Correctly with Multiple Clients	5
Commented Correctly and Throughout	5
GET command implemented correctly – both client-side and server-side	4
ADD command implemented correctly – both client-side and server-side	4
CLEAR command implemented correctly – both client-side and server-side	4
QUIT command implemented correctly – both client-side and server-side	4
Server handles multiple, simultaneous requests on port 5000 via threads and sockets.	4
Client communicates correctly with server via port 5000.	4
Client correctly displays and updates shared canvas using stddraw.	4
Client periodically (every 100ms) polls the server using a GET command.	4
Client correctly captures and handles mouse events for line specification, drawing and transmission.	4
Client correctly captures and handles keyboard events for clearing and quitting.	4
Total	**50**

Example Files

A .zip archive ExamplesForNetworkingLab.zip is included with the lab providing the referenced examples:

- `MAGIC8BALL_tcp_server.py`
- `MAGIC8BALL_tcp_client.py`
- `SingleUserCanvas.py`

Also included are new copies of the following files for drawing capability. Use these instead of prior copies:

- `stddraw.py`
- `color.py`

Communication Protocol

It is important to understand the communication protocol used between the client and server programs. The client and server communicate via a simple text-based protocol transmitted via socket connections. The protocol uses a request-response style of communication, similar to protocols such as HTTP. The client always initiates communication by sending a single line of text to the server. The line consists of a command and parameters, if the command takes parameters. In the case of the ADD command, parameters specify the endpoints of the line segment. After receiving a command from the client, the server performs the appropriate action and sends a single line of text in response. The server always responds with a single line of text, often just simply saying "OK".

Command: `GET`

Parameters: None

Server Response:

A single line of text containing fields separated by spaces. The first field is an integer indicating the number of current Line objects in the Lines collection. Following this are 0 or more 4-tuples giving the details of the lines in the collection. Each 4-tuple consists of 4 floating-point values for the endpoints of the line (x0, y0) - (x1, y1).

Example: the response for a canvas with 2 lines:

```
2 0.65 0.23 0.88 0.5 0.1 0.23 0.33 0.428
```

Command: `ADD`

Parameters: None

A 4-tuple of the form: "`x0 y0 x1 y1`" where (x0, y0) and (x1, y1) are the endpoints of the line segment.

For example, to add a line from the lower-left corner to the upper-right corner:

```
ADD 0.0 0.0 1.0 1.0
```

Server Response:

```
OK
```

Command: `CLEAR`

Parameters: None

Server Response:

```
OK
```

Command: `QUIT`

Parameters: None

Server Response:

```
OK
```

SharedCanvasServer.py

This script is similar to the MAGIC8BALL_tcp_server.py example. It takes a single command line argument, the host that it is bound to. It listens for requests on port 5000, which shall be hardcoded in the script, and starts threads to handle incoming client requests and responses.

It keeps track of and manages the current lines that are on the "shared canvas." It is up to you how to implement storage of the line data. You must ensure that locking is used to prevent concurrency issues because multiple threads will access this data.

When a client connects, the server should print to the console "HELLO:" followed by the client's Socket object information. If the server receives a QUIT command, it responds with "OK". When a client disconnects, print to standard output "GOODBYE:" followed by the client's Socket object.

SharedCanvasClient.py

This script is similar to the MAGIC8BALL_tcp_client.py example. It takes a single command line argument, the host that it will communicate with for access to the server. It sends requests on port 5000, which shall be hardcoded in the script.

The script does the drawing on the client-side. I.e., every client connected to the server will show its own stddraw window to represent the shared canvas. The shared canvas will display the current set of lines as stored on the server.

The client periodically polls the server (approximately every 100ms) using a GET command to obtain the current set of lines. It then redraws (clears first) the contents of the canvas.

All lines drawn by the same client are drawn in black. The start position of a line segment is specified by the user pressing the mouse button on their canvas. The end position of that line is specified when the user again presses the mouse button. After a second mouse button is pressed, an ADD command is sent to the server to add the new line to the shared collection.

The client program supports two commands via the keyboard. The 'q' key causes the client to execute the QUIT command. This should cause an orderly closing of that user's client canvas window. The lines drawn by a user/client should remain in the shared collection even if the client quits. The 'c' key causes the client to execute a CLEAR command. This removes all lines (including those from other clients) from the shared set stored on the server.

Helpful Hints:

Do I need to follow the given specifications? Yes. You must implement methods as described. You may add additional methods if they are helpful to your code.

How do I write a server like this? See the MAGIC8BALL_tcp_server.py example script. It provides an example of a server that can handle multiple, simultaneous client requests.

Do I need to use threads on the server? Yes. You need threads to handle multiple, simultaneous client requests.

Do I need to use locking on the server? Yes. Multiple clients will access the server simultaneously. Without locking the shared variable(s) storing the line data, there will be concurrency issues.

How do I write a client like this? See the MAGIC8BALL_tcp_client.py example script. It provides an example of a simple client that communicates with a server. Realize that example client is simpler than the one to be developed for this lab. It does not draw and it ends after a single request.

Do I need to handle and catch exceptions on the client and server? Yes. Within reason, you should write a robust server and client to prevent exceptions from crashing your scripts.

How do I capture mouse and keyboard events and draw lines using stddraw.py? See the SingleUserCanvas.py example script.

Why doesn't the server display the shared canvas? There is no need for the server to show the canvas. It facilitates communication with clients and storage of the information (lines) that will be drawn. Users use the client to interact with the server and each other and to see the corresponding display. This is similar to how a web server interacts with web clients (browsers).

How should I handle opening and closing socket connections to the server? You should open a connection only when a command is to be sent, send a command/request, get the response and then close the connection. When it is time to send another command/request, open a connection, and repeat the process. For this sort of application, leaving socket connections open when not in use wastes resources. With the approach we take here, a server can handle far more clients that it could if each client kept and always-open connection to the server. This is similar to how web clients and servers work with HTTP, allowing for greater numbers of simultaneous users.

Submit Your Work

After following all instructions above, please submit your Python scripts (`SharedCanvasServer.py`, `SharedCanvasClient.py`) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, and other useful information. Make sure classes, methods and functions are thoroughly commented.

- `SharedCanvasServer.py`
- `SharedCanvasClient.py`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

1. How would you implement circle drawing instead of line segments?

2. How would you implement circle drawing in addition to line segments?

3. How would you implement circle, point, rectangle, etc. drawing in addition to line segments?

4. How would you implement color selection/specification?

5. How would you implement a multi-player game using a client-server network and drawing architecture like the one in this lab?