Luca Imbrogno 160319390
Jonathon Kwon 160593110

CP372 Assignment 1 - Socket Programming

Table of Contents

1 Introduction

The purpose of this program is to provide the user with a functional multithreading client-server network application in Python using the socket API. The server holds a board which manages the placement of notes. The user will specify the size of the board, note colors, host address and port number on initialization. Using the commands, the user is able to post, pin, unpin, and remove notes as well as request to view notes based on certain query parameters.

2 Request Messages - (Assumption → client is requesting data from server)

CONNECT

- Purpose
    - The client is looking to establish a connection with the server via a socket in order to send and receive data. A request is sent from the client to server but also from the server to client when asking for the proper host address and port number.
- Format of request

- In order for the client to create a proper request the client must first enter/input option 1 on the client menu provided:

  1-connect
  2-disconnect
  3-POST
  4-GET
  5-PIN
  6-UNPIN
  7-CLEAR

  After selecting the connect option the client must then input the correct host address (see note below) and port number. The port number should be the same as the one used to initialize the server and board.

  Note: Assume that the host address = "127.0.0.1" and that the user is aware what port number has been used to initialize the board.

  A socket object, on the client side, will then be used to connect to the server leveraging the host address and port number inputted. The host address is read in as a string while the port number is of type integer.

  socket_object.connect((string (host address) , integer (port number)))

  While the server will request

DISCONNECT

- Purpose
  - The client has finished their session and is looking to disconnect from the server so it sends a disconnect request

- Format of request
  - In order for a client to enter a successful disconnect request they must select option 2 off the client menu and input the string "DISCONNECT". This will then be read in by the server and if successful will close the connection and remove the socket related to that specific client.

    self.connected = False
    clients_connected.remove(self)

POST

- Purpose
  - The client is looking to add a note on to the board.
- Format of Request
  - In order for a client to enter a successful POST request there is a specific request

format that must be followed or else an error will be generated.

POST integer(x) integer(y) integer(w) integer(h) "colour" "note message"

The POST request must first include the key word POST, it must then include the coordinate of the lower left corner of the note as well as its corresponding width and height - these must be in the form of integers. Following the numerical values, a corresponding colour and note message must be provided - all of which must be strings. The mistakes/errors of the POST request is explored further in Section 3. Finally, The POST request is digested by the server and will upload/add the note on to the board if the requirements are met.

GET

- Purpose
  - The client sends a request to the server as they are looking to retrieve a note that meets specific parameters - colour, location and substring.
- Format of Request
  - There are 2 GET requests available:

    GET PINS
    GET color=string("some colour")
    contains= (int(x) int(y))
    refersTo="somestring"

The first request, GET PINS, retrieves the coordinate location of all current pins on the board - this request must be in the form of a string. The second request can include any individual or all parameter values

PIN

- Purpose
  - The client sends a request to the server as they are looking to pin a note or group of notes at a specific location
- Format of Request
  - A successful PIN request will be formatted as such:

    PIN int(x),int(y)

    The request will only handle a coordinate value whereby the x and y coordinate

values are integers and represent a valid location on the board.

UNPIN

- Purpose
  - The client sends a request to the server as they are looking to unpin a note or group of notes at a specific location
- Format of Request
  - A successful UNPIN request will be formatted as such:

    UNPIN int(x),int(y)

    The request will only handle a coordinate value whereby the x and y coordinate values are integers and represent a valid PIN location

CLEAR

- Purpose
  - The client sends a request to the server as they are looking to clear the board of all notes.
- Format of Request
  - The client needs to input the string “CLEAR” and the server will then empty/remove all notes from its board.

3 Response Messages

CONNECT

Upon connection to the server the server will send the client an encoded message containing the size of the board and the colors available for notes

self.socket.sendall(str.encode(board_details))

“Board Size: WidthxHeight, Color Options: color A, color B, color C

DISCONNECT

Once disconnect request is made the client will inform the server that it is disconnecting and then the server will send an encoded disconnect message to the client

socket.sendall(str.encode(disconnect_msg))

The server will then close the socket connection to the client and the client will close its socket and output the disconnect messages

"Disconnecting from server…"

"You have been disconnected from the server"

POST

When a valid POST request is made the server will send an encoded success message to the client (or an error message if the note failed to post)

self.socket.sendall(str.encode(error))

GET

There are two types of GET responses. For a GET PINS request, the server will send an encoded message to the client containing a list of the locations of all current pins on the board.

for pin in pin_locations:

self.socket.sendall(str.encode(str(pin)))

"Pin Location: (X,Y)"

"Pin Location: (X,Y)"

…

For the other GET type, the client can request to see all notes that pass the given criteria. The server will send encoded messages containing the list of valid notes and their corresponding messages

for note in valid_notes:

self.socket.sendall(str.encode(note))

"Note 1: note_one_message"

"Note 2: note_two_message"

...

PIN

When a successful PIN request is made the server will send an encoded success message back to the client

self.socket.sendall(str.encode(success_message))

The client will then output the success message

"Pin has been placed"

UNPIN

When a successful UNPIN request is made the server will send an encoded success message back to the client informing the client of the status of the pin

self.socket.sendall(str.encode(success_message))

The client will then output the success message
“Pin has been removed”

CLEAR

When a successful CLEAR request is made the server will send an encoded success message notifying the user of the status of the board

self.socket.sendall(str.encode(message))
“Board has been cleared”

3 Threading & Synchronization Mechanisms

Multithreading

The decision we decided to make to allow for multiple client connections was through the multithreading mechanism as we felt more comfortable with this mechanism over the others. The multithreading capability is implemented on both the server and client side - on the server end a new thread is created and waits to be connected with a new client connection:

new_client_thread = threading.Thread(target= Function “new_client” , args=(socket object,)

The function passed into the thread is “new_client” which waits for a new client connection and then adds the connection to a global array which stores all client connections connected with the server; subsequently the argument that is passed is a socket object. Then all the active client threads are centralized into a single array and are joined. Similarly, on the client side a new thread is created when initially establishing a connection with the server - this thread is used to read in data coming from the server:

data_thread = threading.Thread(target = Function “receive_data” , args = (socket object, True))

The function passed into this thread is “receive_data” which reads in the data sent in from the server and decodes it from bytes to string.

Synchronization Mechanisms

The synchronization primitive we are using are RLocks because we felt they would be able to effectively prevent

any inconsistent output and we also wanted to avoid unwanted blocking by leveraging their unique property - that being only a different thread will be blocked opposed to the current thread itself. We implemented the RLocks in our “run” function on the server side:

```
def run(self):
    lock.acquire()
    self.socket.sendall(str.encode("You now have access to
edit the board and interact with the server"))
    ….
    lock.release()
```

Because the dictionary (our selected data structure) is a global structure holding all of the client notes we wanted to avoid the situation where multiple clients could be editing and/or accessing the same note or pieces of data - so we made the design decision that only one client could edit the board/note dictionary and interact with the server at a time. To be clear, this does not mean that only one client can connect with the server at a time - the server can still accept multiple client connections however, only one client (the client that connected the earliest - FIFO) can edit and request data from the board/server at a time. That means all the other connected clients will receive a success message once they connect to the server but will not be able to interact with the server until the client ahead of them has finished. Once a client has finished their session (meaning they have disconnected) a message will be displayed informing the next client that they now have access to edit the board (note dictionary) along with the client menu being outputted as well.

```
lock.acquire()
self.socket.sendall(str.encode("You now have access to edit
the board and interact with the server"))
```

The RLocks implemented in the “run” function on the server side will allow for multiple client connections but will ensure that only one client, in this case one unique thread, can edit the data structure that is shared with all other clients.

5 Error Handling

Menu

Client side error handling. If the user enters a non-integer value or some value outside of menu options they will receive: “Cannot accept non-int types- must select from client menu”

CONNECT

Client side error handling. If the user tries to execute any other commands before connecting to the server the user will be prompted with "Must connect with server first (1-connect). Enter 1 to start connection with server.”

If the user enters incorrect or invalid host address or port number they will be prompted with: "Error - please check your host address and port #”. Additionally, they will be prompted to re-enter the host and port number.

DISCONNECT

Error handling is done on the server side. Invalid ‘DISCONNECT’ command will result in the error message “Invalid Request”

POST

Error handling is done on the server side. Error messages are as follows:

- If the user inputs some unknown string
  “Invalid statement”
- If the user does not enter all the parameters required
  “Incomplete statement”
- If the user enters any command other than POST
  “Key word POST is missing”
- If the user tries to post a note outside the board dimensions
  “Cannot post note outside board dimensions - please try again”
- If the user tries to use an invalid colour
  “Note colour not acceptable”
- If the user does not enter a message with the note
  “Cannot have empty note message - please include a message”
- If the user enters a message that has already been posted
  “Note already exists”

GET

Error handling is done on the server side. Error messages are as follows:

- If the user enters some unknown string
  "Invalid statement"
- If the user does not enter all the necessary parameters
  "Incomplete statement found"
- If the user enters GET PINS with no pins currently on the board
  "Error no pins available"
- If the user does not use the proper get parameters with '=' signs
  "No proper GET parameters found"
- If the users parameters do not match any notes
  "No notes found with those parameters"

PIN/UNPIN

Error handling is done on the server side. Error messages are as follows:

- If the user enters some invalid type in their request
  "Invalid Statement"
- If the user does not enter the proper parameters
  "Incomplete statement found"
- If the user enters any command other than PIN or UNPIN
  "Key word PIN (UNPIN) is missing"
- If the user tries to place pin outside board dimensions
  "Cannot place pin outside board dimensions - please try again"
- If the user tries to remove a pin from a location where there is no pin currently placed
  "There are no pins currently at this location - please try again"

CLEAR

Error handling is done on the server side. Error messages are as follows:

- If the user enters any command other than CLEAR
  "Key word CLEAR is missing"

6 Server Shutdown

Following the disconnection of any client's session the server will prompt the operator with an input message to which they must respond to:

quit = input("Do you want to close the server? If yes enter - close, if no enter - no : ")

If the operator types in "close" the server will proceed to shutdown and if they enter "no" then the server will continue with operations.