Created: February 1, 2020

Authors: Amelia Kang, Faiq Ahmed

Computer Networks Assignment 1

Socket Programming in Python

Application-Layer Protocol

Abstract

This documentation introduces an application-layer protocol implemented in Python using Socket API. The application functions as a note board where multiple users can post notes onto the board, view all the posts, pin, unpin, and clear unpinned messages.

Table of Contents

1) Introduction

This application simulates a bulletin note board where multiple clients can open a TCP connection to the server and post and read notes from it. The notes can have different features including position, size, colour, content, and status as pinned/unpinned.

Note is defined as a Python dictionary object: {x, y, width, height, color, message, is_pinned}

The main functionalities of the application include:

1) CONNECT the client and server
2) POST notes
3) GET all notes, pinned notes, and notes with specific colour, content, location
4) PIN/UNPIN notes from the board
5) CLEAR any unpinned notes
6) DISCONNECT the client and server

2) Synchronization Mechanism

Multi-threading mechanism is used in this application because this application is designed to allow multiple clients to connect to the server and send requests simultaneously, and multi-threading is a simple way to let it achieve that purpose. Moreover, due to its simple structure and functionalities, this application does not require much CPU power to operate, thus a single process with multiple threads is fairly enough; in other words, no multi-processing is needed.

3) Error handling

Most error handlings are implemented using conditions and try-except block. During the occurrence of any errors, both error messages suggested by the client/server side program and the actual run-time error message caught by the machine are sent to the client side through print function. Generally, if user inputs any requests that are not in a list of accepted request codes, client side will reject the input and ask the user to try again; client side deals with user-fault errors such as formatting errors and insufficient argument errors; server side deals with errors that comply to the rules in this application (pinning already pinned notes) and socket errors (index out of bounds), and sends them back to the user as a server response.

a) **CONNECT**
   i) **Client Side Check:**
      (1) Already connected: a message indicating that the client is already connected to the server will be provided to the user.
      (2) Wrong port number (i.e. not a number): client side will send an error message "Incorrect port given. Port must be a number e.g. 9090", ask the user to try again, and terminate client.py.
      (3) Not enough info given: if the user does not provide enough information (missing host or port) or provides it in a wrong format (e.g. using comma as delimiter), client side will send error message "Insufficient server information given" with an example input, and ask the user to try again.

   ii) **Server Side Check:**
      (1) Wrong host and port information: Error message "Connection cannot be established" will be passed to the client side, along with the run-time error message generated by the socket.

b) **DISCONNECT**
   i) **Client Side Check:**
      (1) Already disconnected or no existing connection: message "Client has not opened a connection or connection already closed" will be sent to user, and client side will terminate.

   ii) **Server Side Check:**
      (1) Connection cannot be closed: any error regarding disconnecting the client will be caught by except block and passed to the client side, to be printed to the user.

c) **POST**
   i) **Client Side Check:**
      (1) No post request content provided: "No input provided. Please try again"
      (2) Not enough request content: user might not have provided enough parameters required by the format of a note. Each note must have at least 4 parameters: x, y, width, height, message, colour and pinning status are optional. Error "Insufficient params in the request" will be sent to user.

(3) x, y, width, height are not numbers: Error "Invalid request input: the first 4 inputs must be numbers." will be shown to the user.

(4) Width and/or height are 0: "Width and height of the note must be greater than 0. Please try again" will be shown to the user.

ii) Server Side Check:

(1) If defined note position is outside the board size, or any side of the note is longer than the board length, error "Note does not fit on the board, please resize/reposition to fit in. (500x600)." will be raised and sent to the client side along with the dimension of the note board.

(2) Given negative positions and sizes: will be converted to positive without acknowledging the user.

d) GET

i) Client Side Check:

(1) No GET input given: "No input provided" and ask the user to try again.

ii) Server Side Check:

(1) When GET with conditions color=red contains=7,7 refersto=fred for example: if any condition does not have '=' in it, the server logic will continue to the next item, but adding error "Invalid pair:" with the invalid item to the response to be passed to the client side.

(2) Invalid coordinates provided to contains=: Program skips to the next item, adding error "Invalid coordinates/formatting. Please follow format x,y (no space)" to the response to be passed to the client side.

(3) Keyword not found in accepted commands ['color','contains','refersto']: error "Invalid GET parameter" will be sent to client side with the item.

(4) No qualified notes found: if no notes found after searching through the given criteria, server returns response "No qualified notes found" to client side.

e) PIN/UNPIN

i) Client Side Check:

(1) If coordinates are not separated by a comma, "Invalid format of PIN/UNPIN parameter. Example: 6,7" will be printed to the user.

(2) This function does not allow more than one pair of coordinates given. If so happens, a message with "invalid format" will be printed to the user.
(3) More than 2 coordinates are provided (e.g. 7,7,8): "Invalid coordinates entered. Please retry and enter 2 digits" will be printed to the user.

ii) Server Side Check:

(1) Already pinned note: "Note at xy already pinned (not pinned)" with coordinates will be passed to the client side as response.
(2) If input coordinates do not have length 2 or is not separated by a comma, "Invalid format of PIN/UNPIN instruction. Please use format PIN/UNPIN X,Y and try again" will be sent as a response.

f) CLEAR

i) Not much can go wrong with CLEAR request, thus no error handling is implemented as of this point.

4) Request message sent by client (case insensitive)

User can choose to either provide the entire request or give request code (POST, GET, etc.) and wait for instructions then give the rest required parameters.

a) "CONNECT": connect to server

i) If user provides full request: format: connect host port. Example: connect localhost 9090

ii) If not, instructions will be provided:

(1) Connect
(2) "Enter server IP and port separated by space:" localhost 9090

b) "DISCONNECT": disconnect to server

i) Disconnect

c) "POST": post message to server

i) If user provides full request: format: POST X Y width height (color) message (status); color and pinning status are optional, if not provided, both will be set to default. Example: POST 6 5 10 12 red hello world pinned

ii) If no:

(1) POST
(2) "Request requires: coordinates, width, height, color, "message" and status, separated by space. Example: 6 6

5 5 red Pick up Fred from home at 5 pinned. Note that negative values will be converted to positive. Enter your POST message:” 6 6 5 5 red Pick up Fred from home at 5 pinned

d) **GET ALL/PINS/color=color contains=coordinates refersto=value: get notes by criteria**
   i) GET all -- gets all messages posted
   ii) GET PINS -- gets all pinned messages
   iii) GET color=red contains=7,7 refersto=fred -- get red notes that contain word “fred” at coords 7,7

e) **PIN/UNPIN x,y: pin or unpin message(s) on the note board**
   i) Full request: PIN/UNPIN 7,7 - pin/unpin the notes that have coords 7,7
   ii) Code only:
      (1) PIN/UNPIN
      (2) “Example 7,7. Enter a pair of coordinates separated by comma:” 6,6

f) **CLEAR: clear any unpinned messages off the note board**
   i) clear

5) Response message sent by server

a) **CONNECT**
when the client requests a **connection** with the server and if the server is able to connect, the response messages sent by the server are in the form of a string to let the client know of the colours of the notes, and the dimensions of the board. In addition to sending a response message to the client, the server also prints a few messages stating the IP and the port number of clients connected and the number of currently running client threads.

   i) Once connected, server sends “Available colors for notes:” and “Board size:” with colors and board size defined upon starting the server to the client.
   ii) If fails, server will insert the exception into the response and send it back to the client.

b) **DISCONNECT**
   i) If successful: respond “Disconnected client” to client.
   ii) If unsuccessful: respond the client with the exception caught.

c) **POST**

  i) If successful: "Note added to the board successfully". Apart from the response message for the client, the server also prints the parameters it reads for the post request in the server window.

  ii) If no: error messages defined by the server logic and/or the exception messages caught by the machine or socket API will be sent to client side (error handling).

d) **GET**

  i) GET ALL: all the notes will be fetched and sent to client.
  Example:
```
All notes:
7,7 - red - hello world - True
3,4 - yellow - hello internet - False
6,8 - blue - hello perfect assignment - True
```

  ii) GET PINS: get all the pinned messages and send to client. If no pinned notes found, blank will replace the coordinates below.
  Example:
```
All pins locations:
6,7
4,4
```

  iii) GET color= contains= refersto= : get qualified notes corresponding to the query
    (1) If notes exist:
      "Notes that satisfy criteria color=red refersto=fred:
      5,6 - red - pick up fred"
    (2) If no notes are found:
      "No qualified notes found."

e) **PIN/UNPIN**

  i) If note(s) found, server will respond:
  "x note(s) pinned" or "x note(s) unpinned" if command is UNPIN, x is the number of notes pinned or unpinned.

  ii) If no note found:
  "No note found at coordinate x,y"

f) **CLEAR**

  i) The server responds the client with "x unpinned note(s) removed", x as the count of removed notes. The number could be zero if all the notes are pinned.

6) Contributors

Amelia Kang

Faiq Ahmed