Computer Networking A1

Request for Comments: “Post it Board a Client/Server Application”

Duncan Wilson - 150322290

Pranav Patel - 150380910

Contents

[1 Introduction 2](#_Toc536373980)

[1.1 Purpose 2](#_Toc536373981)

[1.2 Requirements 2](#_Toc536373982)

[1.2.1 Client file 2](#_Toc536373983)

[1.2.2 Server file 2](#_Toc536373984)

[1.3 Terminology 2](#_Toc536373985)

[2 Protocol 3](#_Toc536373986)

[2.1 Client side 3](#_Toc536373987)

[2.2 Server side 3](#_Toc536373988)

[3 Message Format 4](#_Toc536373989)

[4 Synchronization Policy 5](#_Toc536373990)

[5 Error Handling 5](#_Toc536373991)

[5.1 Client side 5](#_Toc536373992)

[5.2 Server side 5](#_Toc536373993)

[6 Border Cases 6](#_Toc536373994)

# 1 Introduction

## 1.1 Purpose

The purpose of the client-server application is to support the client’s ability to store items called “notes” on a server which represents a board, and to request these “notes” with certain properties and commands. The board will be a way of information storage for the clients with all the posts on it and provide a way to add and remove the posts.

## 1.2 Requirements

For the completion of the program, two java files are required. One for the client and one for the server.

### 1.2.1 Client file

From the client java file:

* Requests to and connections with the server
* Communication via String messages from the list of possible commands discussed in 3. “Message Format”
* Ability to disconnect from server

### 1.2.2 Server file

From the server java file:

* Able to handle multiple connections
* Utilize data structures to handle data inputs from client
* Process requests from client, further discussed in 2. “Protocol”

## 1.3 Terminology

Here is a list of terms with their definition/purpose:

* Notes – contains information in String format
* Post – the request from client to send a note with corresponding parameters to server
* Board – the “data structure” and “GUI” which will hold the notes from a client request
* Data Structure – storage for the data and properties of the notes
* Graphical User Interface – referred to as GUI – visual representation of the notes stored and way for client to send requests to server through button clicks
* Requests – an action client wants the server to complete, format will be discussed in 2. “Protocol”
* String – input format for the notes
* Pin/Unpin – actions client can send to server for specified notes
* Get – action client can send server for specified notes
* Clear – action to remove notes which are not pinned
* Disconnect – disconnect from server
* Port 4554 – the socket port number which will be used for connection to server
* Client GUI – the graphical user interface for client to interact with server
* <> - represents places where input from user is required, it will be inserted from text field in GUI

# 2 Protocol

## 2.1 Client side

From the client side there will be two main components. The first one is the connection and communication to the server and the second one will be the graphical user interface which will allow user to interact with client to send requests to server.

The client will request a connection to the server on port 4554 and will receive communication upon successful connection. From there the connection will be given a number which will be used to identify what client is sending requests to the server. The GUI will allow the user to interact with client and compose a message to send to server. A String message will be created from the GUI which will be sent to sever for processing.

The GUI will be created from classes in the client file to allow multiple frames. The GUI will have a top header display for information on IP, port, connection, and other status field. There will be buttons on the left side of GUI to allow connection and disconnection for server, post to server and pin/unpin posts. The center body of GUI will allow user to insert text to be sent to server and below the input field will be the display of the results received from server. The GUI will be a simple 2d interface.

Refer to 5. “Error Handling” to see how the error handler will function on the client.

## 2.2 Server side

From the server side there will be three main components. The first one will be to create connection to client and handle input communication via String message on command line. The second will be to process and keep track of the posts and board from start to finish. The third one will be communicating the results back to the client.

The server will be multithreaded using safe thread classes in Java and it will connect to the client on socket port 4554.

The input message will be of type String and contain many required parameters please refer to 3. “Message Format” for detail on the valid formats. Incorrect formats will be handled by the error handler please refer to 5. “Error Handling”. The information from the message will be placed in the correct data structures and utilized when needed according to the requests.

2d arrays will be the data structures used to track the data and properties of the posts and board, they will be active for the running connections. They will store data according to the message format for better readability.

The states of each level will be logged where possible, specifically when receiving the message from client and processing a request. The results will be sent back to the client via String message from the server.

# 3 Message Format

The client and the server will take String inputs in lowercase format. All inputs no matter what letter case will be converted into lowercase.

Here are a list of possible requests and an example of each:

1. POST <note>
   1. There will be a button on the GUI to post the message in the text field
   2. The button will add a “post” string to the start of the text field and send to server
   3. <note> will consist of coordinates of lower left corner (2 numbers, x and y), width, height, color, and message. This will all be typed by the user in the GUI
      1. Example “2 2 3 3 red message here
      2. There will only be 1 word or number to a parameter only the 6th parameter which is the last one will contain multiple words or numbers (part of message)
2. GET <request>
   1. There will be a button on the GUI to send a get request
   2. <request> will have either 1 word or multiple
      1. The 1-word format will but <request> = “pins”, meaning server has to send list of current pinned posts
      2. The multiple word format will include, in this order, color=<color> contains=<data> refersTo=<string>
         1. There are 3 parameters, not all have to be inputted. Where a parameter is not inputted it will be assumed by program to not use that criteria when processing the request for specific posts
3. PIN/UNPIN <data>
   1. There will be buttons for pinning and unpinning posts which will read the input text for the <data> part of the message to send to the server
   2. The <data> part will be formatted to be coordinates (2 numbers, x and y) of where the pin will go, or where pin will be removed from
4. CLEAR and DISCONNECT will have corresponding buttons on GUI which will send a request to server
   1. The string message communicated to server will be lowercase words

# 4 Synchronization Policy

Thread safe classes will be implemented. Operations done on the board by different requests from clients will be concurrent, but will not all be completed at the same time per say. There will be a queue that tracks and monitors multiple requests. Requests affecting the same posts or multiple posts which are active in other requests will be paused until free to continue. Certain processes will be marked as synchronized processes and they will only process 1 by 1. Multiple requests to those functions will pause that thread.

# 5 Error Handling

## 5.1 Client side

Potential errors in client:

* Error in connection to server – make the process to connect to server easy with only input for IP address. If incorrect IP provided connection not created and error handler will loop to allow user to input IP until it is correct, or they cancel.
* Incorrect inputs – if connection is made then, inputs will be validated on the list of inputs that are possible and if it does not meet the message protocol server will send request for client to send different inputs or disconnect
  + Specifically, from the GUI each button on it will have their own data validation. For example, the PIN/UNPINN buttons will require coordinate inputs when pressed and if the requirement is not met the client will prompt user to input values again.
* Multiple frames causing crash or freeze – have a force close on an event of this type and disconnect server

## 5.2 Server side

Potential errors in server:

* Error in connection to client – display error messages accordingly and advice user to check socket number if no other issues discovered.
* Threading issues – have proper processes and thread safe Java classes but on error will display error message accordingly and try to go back to state before the error causing request was sent
* Incorrect inputs – inputs that are valid on client side might not be valid on server like posting outside of board or negative values
  + Server will display proper error message and request client send another message with proper inputs

# 6 Border Cases

Here are potential cases and their solutions for our implementation:

* Unpin a blank location: display message stating no pins to remove, and nothing changed
* GET request without posts on board: display message stating no posts available
* Server shut down unexpectedly: client displays disconnection message
* Client crashes: server disconnects client
* What happens on long string message: have warning not to go over character limit, cut of characters that go over limit.
* What happens if note with empty string posted: internally assign a value stating it is empty and proceed posting as a blank note
* What happens if note with empty string requested: if the post exists it will be marked as empty and will be retrieved and its properties like color will be displayed
* Clear board with no posts: nothing needs to happen, proper output messages will be displayed upon completion of process