CP372 – Assignment 3 Documentation

Dallas Fraser (110242560)
George Lifchits (100691350)

April 4, 2016

1 Introduction

The following document describes `PaintProtocol/1.0`, a protocol wherein a server enables clients to interact with a whiteboard. Client and server communicate via a stream socket connection. The protocol supports multiple clients.

1.1 Terminology

This document uses the following terms, which are originally defined in [1]:

connection
: A transport layer virtual circuit established between two programs for the purpose of communication.

client
: A program that establishes connections for the purpose of sending requests

server
: An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the programs capabilities in general. Likewise, any

server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

reader
A program which dispatches a GET request. The reader must acquire a read lock.

writer
A program which dispatches a REMOVE or SET request. The writer must acquire a write lock.

2 Protocol

The protocol has three different type of requests: GET, POST, and DELETE. Each request interacts with a list of synonyms.

2.1 Message Format

2.1.1 GET Request

A GET request is used to find all the non-white points on the whiteboard. The following describes the format of the request and its response message format.
Request: The request follows the following format:

```
GET
```

Response: The response follows the following format:

```
PaintProtocol/1.0 STATUS-CODE MESSAGE
Content-Type: text/html

LIST OF POINTS
END
```

where

`STATUS-CODE` result of the request. See section 2.2 (Integer)

`MESSAGE` corresponding HTTP message for the status code (String)

`LIST OF POINTS` all the points that are on their own lines and are in the following format: `xx yy rr:gg:bb` (String)

`END` a blank empty new line

Example:

```
GET

PaintProtocol/1.0 200 Successful
Content-Type: text/html

0 0 00:255:255
```

2.1.2 POST Request

A POST request is used to add a point(s) to the whiteboard. Each point should be one its own line. The following describes the format of the request and its response message format.

Request: The request follows the following format:

```
POST
x y rr:gg:bb
...
END
```

where

`x` the x coordinate of the point (int)

`y` the y coordinate of the point (int)

`rr:gg:bb` the color of the point (string)

`END` a blank empty new line or END (String)

Response: The response follows the following format:

```
PaintProtocol/1.0 STATUS-CODE MESSAGE
Content-Type: text/html

x y rr:gg:bb
...
```

where

`STATUS-CODE` result of the request. See section 2.2 (Integer)

`MESSAGE` corresponding HTTP message for the status code (String)

`MORE DETAILS` any additional details about the request results (String)

`END` a blank empty new line or END (String)

Example:

```
POST
0 0 255:255:255
0 1 255:255:255
END

PaintProtocol/1.0 201 Created
Content-Type: text/html

0 0 255:255:255
0 1 255:255:255
```

2.1.3 DELETE Request

A DELETE request is used to remove a point(s) from the whiteboard. The points will be deleted from the shared whiteboard.
Request: The request follows the following format:

```
DELETE
x y
x y
END
```

where

`x` the x coordinate of the point (int)

`y` the y coordinate of the point (int)

`MESSAGE` corresponding HTTP message for the status code (String) (see section 2.2)

`END` a blank empty new line or END (String)

Response: The response follows the following format:

```
PaintProtocol/1.0 STATUS-CODE MESSAGE
Content-Type: text/html

x y
x y
END
```

where

`STATUS-CODE` result of the request. See below (Integer)

`MESSAGE` corresponding HTTP message for the status code (String)

`MORE DETAILS` on why server was unable to complete the request (String)

`END` a blank empty new line or END (String)

Example:

```
DELETE
0 0
0 1
END

PaintProtocol/1.0 200 DELETED
Content-Type: text/html

0 0
0 1
```

2.1.4 Sync Response

A Sync response is used when a bad request has been made and requires all the whiteboards to sync to the same points. This prevents whiteboards from becoming out of sync. The following describes the format of the response message format.

Response: The response follows the following format:

```
PaintProtocol/1.0 STATUS-CODE MESSAGE
Content-Type: text/html

x y rr:gg:bb
x y rr:gg:bb
END
```

where

`STATUS-CODE` result of the request. See below (Integer)

`MESSAGE` corresponding HTTP message for the status code (String)

`END` an blank empty new line (String)

Example:

```
PaintProtocol/1.0 200 Sync
Content-Type: text/html

0 0 255:255:255
0 1 255:255:255
```

2.1.5 QUIT Request

A QUIT request is used to disconnect from the server. The following describes the format of the request. There is no response message from the server.

Request: The request follows the following format:

```
QUIT
```

2.2 Status Codes

2.2.1 GET Request

Status Code	Message	Description
200	Successful	The GET was successful
400	Bad Request	Request parameters are either malformed or omitted
404	Not Found	Signals the given word was not found
408	Request Timeout	The request could not be completed on time
500	Internal Server Error	The server encountered an unexpected error while processing the request

2.2.2 POST Request

Status Code	Message	Description
201	Created	The point(s) was(were) added
400	Bad Request	Request parameters are either malformed or omitted
408	Request Timeout	Signals the request could not be completed on time
500	Internal Server Error	The server encountered an unexpected error while processing the request

2.2.3 DELETE Request

Status Code	Message	Description
200	Deleted	Signals the point(s) was(were) deleted
400	Bad Request	Request parameters are either malformed or omitted
404	Not Found	Signals the given word was not found
408	Request Timeout	The request could not be completed on time
500	Internal Server Error	The server encountered an unexpected error while processing the request

2.2.4 Other status codes

Status Code	Message	Description
405	Method Not Allowed	The provided request verb is unsupported

2.3 Synchronization Policy

The server uses a Read and Write Lock to maintain synchronization for the whiteboard. Any thread processing a `GET` request will have to acquire a read lock to retrieve the Points and acquire a write lock to `DELETE` or `POST` Points. There can be many readers concurrently, but only one writer. A write-lock request is prioritized while waiting to avoid starvation. It is assumed there will be more readers than writers. The writer will wait until all current readers are done and then will gain access. Any subsequent readers will have to wait for the current writer to release its lock. If a lock cannot be acquired after three seconds, the request timeouts, which yields a response with status code 408.

The updates to all the other clients are synchronized and one thread sends the all the updates. There is a lock on the socket to ensure synchronization of updating thread and thread handling the requests. Finally, the update thread uses a queue to communicate with the other threads and processes them on a first come first serve basis.

2.4 Exception Handling

There are many exceptions that could occur on the server side. The Exceptions are handled while processing a request, and if an exception is detected, an appropriate status code and message is returned to the client. Exception handling is implementation specific, but many classes of exception map logically into status codes of this protocol. In such a case, the 404 error would be appropriate to signal that the word was not found in the server database.

Any unexpected exceptions which are raised during the server processing of a request should cause the server to return a status 500: internal server error to the client.

2.5 Handling Edge Cases

Edge cases are handled by ensuring all clients are synchronized. A sync request will be sent to all clients. This will notify them when a request is made but is unable to be handled. This is a catch all solution which ensures all clients are synchronized.

Note: specific edge cases have not been discovered and this policy has not been implemented.

References

[1] Berners-Lee, Tim; Fielding, Roy T.; Nielsen, Henrik Frystyk. *Hypertext Transfer Protocol – HTTP/1.0.* IETF. RFC 1945.