Computer Networks and Security

2506ICT

Assignment 2

Level 3 Implementation

23/10/16

Aidan McMillan

S2946192

aidan.mcmillan@griffithuni.edu.au

CONTENTS

INTRODUCTION

This report details the program created using Java which provides the function of a very simple proxy cache. The program implements all required features outlined in **all three levels** of implementation as far as I am aware of.

NOTE: This is a very basic implementation and therefore there are some things that need to be outlined. IMPORTANT details are below.

- Loading too many pages at once will 'overload' the port/socket and cause the program to stall. In this case CTRL+C is the only way to stop the program.
- Simple Pages work best (sqlzoo.net or an image on imgur.com).
- If trying to load a page without the cache disabled/cleared the program will get errors and/or crash. It is recommended to use Step 2 below in Google Chrome to prevent this.
- Validation is minimal to null. If you enter the wrong data or the client sends the wrong request the best that will happen is the program will continue to the next request, the worst is a crash.
- Errors are in the program and display frequently – this doesn't seem to affect the function of the program however.
- Object size is currently set to 1MB.
- Please contact me if a major error is preventing an important function of the program – I understand how it works and why it behaves like it does quite well.

COMPILING AND RUNNING

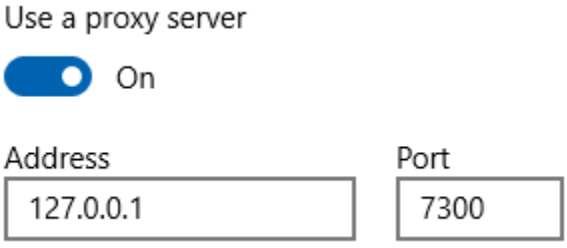
The program can be run simply by navigating to the executable .jar file using command prompt and executing the following command:

Java –jar proxyCache.jar [port]

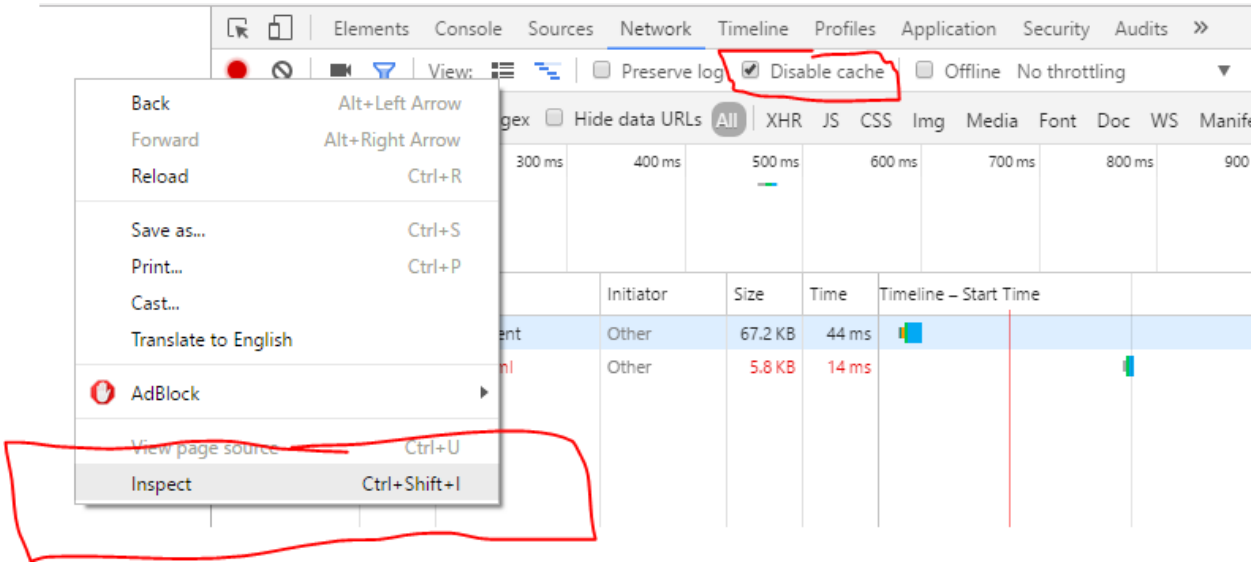
```
C:\Users\Aidan\Desktop>java -jar ProxyCache.jar 7300
Waiting..
```

In order for the program to work properly, however, you will need to follow the following steps:

1. Set your internet settings to use a proxy with these details:
    - Address – 127.0.0.1 (localhost)
    - Port – the port entered above.



2. You must also clear your cache every time you load a cache, or if using Google Chrome, you can keep it disabled using inspect element. This is the easiest method.
   MUST BE KEPT OPEN ON THE PAGE YOU ARE TESTING.



LOGIC AND FLOW

The program makes use of three class file; ProxyCache, HttpRequest, and HttpResponse. Both request and response are used as objects while ProxyCache acts as the controller and main method.

Upon start of the program the main method reads in a given port number and waits. Upon receiving a GET request from the client the handle() method is called and the function of the program begins.

First off the various values are initialized. This includes the creation of a ServerSocket and Hashmap. The HashMap acts as a cache of HttpResponse Objects with key value of a GET request.

The GET Request sent by the client is compared to this hashmap to see if it is in the cache. If it isn't then it is forwarded through to the host. If it is then headers are appended to the request to determine whether the cached version is fresh (If-Modified-Since and If-None-Match).

The response is received, and if it is "Not Modified 304" then the cached version is forwarded to the client. Otherwise the response is placed into the cache (if already there will overwrite) using the key value of the GET request.

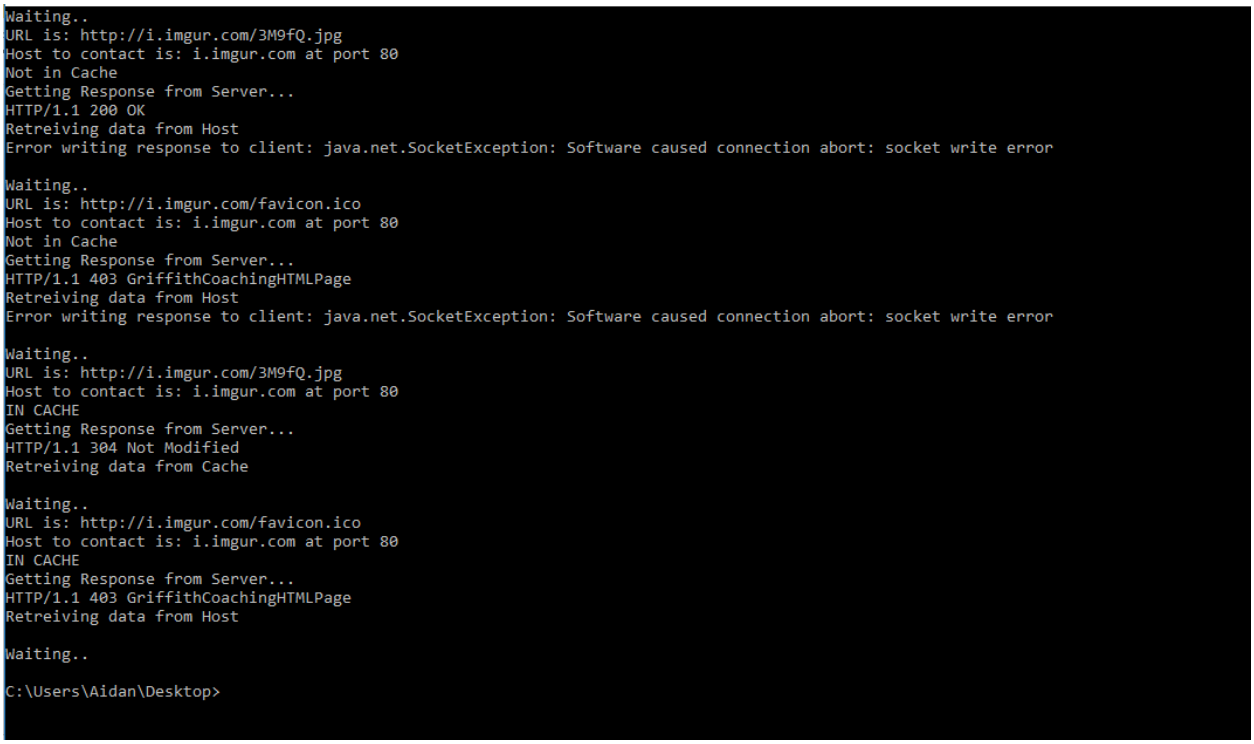
After this the program goes back into a read state while waiting for other requests from the client.

RUNTIME EXAMPLE

Below is a simple example of the flow of the program. As can be seen the webpage is being accessed twice, with the favicon being requested as well as the content each time. (Ignore error – doesn't affect response being forwarded to the client).

In the second iteration it can be seen that the response is found existing "IN CACHE". The program then appends the headers corresponding to modification onto the GET REQUEST and sends it the host. Upon receiving a response of 304 it loads the item from the cache instead. If it receives anything already in the cache that has been modified it will forward the response to the program.

USE CTRL+C to exit program.



SOURCE CODE

PROXYCACHE

```
package cns;

import java.net.*;
import java.io.*;
import java.util.*;

public class ProxyCache {

   private static int port;
   private static ServerSocket socket;
   private static HashMap<String, HttpResponse> cache;

   public static void init(int p) {
                        port = p;
                        cache = new HashMap<String, HttpResponse>();
                        try {
                           socket = new ServerSocket(p);
                        }
                        catch (IOException e) {
                           System.out.println("Error creating socket: " + e);
                           System.exit(-1);
                        }
   }

   public static void handle(Socket client) {
                        Socket server = null;
                        HttpRequest request = null;
                        HttpResponse response = null;

                        // Read request
                        try {
                           BufferedReader fromClient = new BufferedReader(new InputStreamReader(client.getInputStream()));
                           request = new HttpRequest(fromClient);
                        } catch (IOException e) {
                           System.out.println("Error reading request from client: " + e);
                           return;
                        }

                        //Check if request is stored in cache
                        String requestLine = request.toString();
                        if(cache.containsKey(requestLine)) //Check if GET request is in cache
                        {
                                 //Send GET request with last-modified header
                                 System.out.println("IN CACHE");
                                 response = cache.get(requestLine);              //Get response from cache
                                 request.checkModified(response.getModified());      //append modified to header
                                 try {
                                    server = new Socket(request.getHost(), 80);
                                    DataOutputStream toServer = new DataOutputStream(server.getOutputStream());
                                    toServer.writeBytes(request.toString());
                                    toServer.flush();
                                 } catch (UnknownHostException e) {
```

```
                System.out.println("Unknown host: " + request.getHost());
                System.out.println(e);
                return;
            } catch (IOException e) {
                System.out.println("Error writing request to server: " + e);
                return;
            }
}
else{
            //Send Normal GET request
            System.out.println("Not in Cache");
            try {
                server = new Socket(request.getHost(), 80);
                DataOutputStream toServer = new DataOutputStream(server.getOutputStream());
                toServer.writeBytes(request.toString());
                toServer.flush();
            } catch (UnknownHostException e) {
                System.out.println("Unknown host: " + request.getHost());
                System.out.println(e);
                return;
            } catch (IOException e) {
                System.out.println("Error writing request to server: " + e);
                return;
            }
}

//Get Response from server
try {
   DataInputStream fromServer = new DataInputStream(server.getInputStream());
   System.out.println("Getting Response from Server...");
   response = new HttpResponse(fromServer);
   server.close();
} catch (IOException e) {
   System.out.println("Error writing response to client: " + e);
}

//Check response to see if unmodified
if(response.checkModified()){
            System.out.println("Retreiving data from Cache");
            response = cache.get(requestLine);                                    //Get Data from cache
}
else{
            System.out.println("Retreiving data from Host");
            cache.put(request.toString(), response);                    //Get Data from response
}


//Write data to client
try {
   DataOutputStream toClient = new DataOutputStream(client.getOutputStream());
   toClient.writeBytes(response.toString());
   toClient.flush();
   toClient.write(response.body);
   toClient.flush();
   client.close();
} catch (IOException e) {
```

```
                    System.out.println("Error writing response to client: " + e);
                }
                System.out.println();
        }

/* ------------------------------------------------- */


    /** Read command line arguments and start proxy */
    public static void main(String args[]) {
        int myPort = 0;

        try {
            myPort = Integer.parseInt(args[0]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Need port number as argument");
            System.exit(-1);
        } catch (NumberFormatException e) {
            System.out.println("Please give port number as integer.");
            System.exit(-1);
        }

        init(myPort);

        /** Main loop. Listen for incoming connections and spawn a new
         * thread for handling them */
        Socket client = null;

        while (true) {
            try {
                System.out.println("Waiting..");
                    client = socket.accept();
                    handle(client);
            } catch (IOException e) {
                    System.out.println("Error reading request from client: " + e);
                    continue;
            }
        }
    }
}
```

HTTPREQUEST

```
package cns;

import java.io.*;
import java.net.*;
import java.util.*;

public class HttpRequest {

   final static String CRLF = "\r\n";
   final static int HTTP_PORT = 80;
   String method;
   String URL;
   String version;
   String headers = "";
   private String host;
   private int port;

   //Created using clients output socket
   public HttpRequest(BufferedReader from) {
                  String firstLine = "";
                  try {
                     firstLine = from.readLine();
                  }
                  catch (IOException e) {
                     System.out.println("Error reading request line: " + e);
                  }

                  String[] tmp = firstLine.split(" ");
                  method = tmp[0];
                  URL = tmp[1];
                  version = tmp[2];

                  System.out.println("URL is: " + URL);

                  if (!method.equals("GET")) {
                     System.out.println("Error: Method not GET");
                  }
                  try {
                     String line = from.readLine();
                     while (line.length() != 0) {
                           headers += line + CRLF;
                           /* We need to find host header to know which server to
                            * contact in case the request URL is not complete. */
                           if (line.startsWith("Host:")) {
                              tmp = line.split(" ");
```

```
                        if (tmp[1].indexOf(':') > 0) {
                                String[] tmp2 = tmp[1].split(":");
                                host = tmp2[0];
                                port = Integer.parseInt(tmp2[1]);
                        } else {
                                host = tmp[1];
                                port = HTTP_PORT;
                        }
                    }
                    line = from.readLine();
                }
            }
            catch (IOException e) {
                System.out.println("Error reading from socket: " + e);
                return;
            }
            System.out.println("Host to contact is: " + host + " at port " + port);
    }

    //Return Host
    public String getHost() {
            return host;
    }
    //Return port
    public int getPort() {
            return port;
    }
    //Return URL
    public String getURL() {
            return URL;
    }

    //Append Last-Modified to headers
    public void checkModified(String[] modified){
            if(modified.length > 1)
            {
                    headers += "If-None-Match: " + modified[1] + CRLF;
            }
            headers += "If-Modified-Since: " + modified[0] + CRLF;
    }

    //Convert Request into String
    public String toString() {
            String req = "";
            req = method + " " + URL + " " + version + CRLF;
            req += headers;
            req += "Connection: close" + CRLF;
```

```
                req += CRLF;
                return req;
    }
}
```

HTTPRESPONSE

```
package cns;

import java.io.*;
import java.net.*;
import java.util.*;

public class HttpResponse {
    final static String CRLF = "\r\n";
    final static int BUF_SIZE = 8192;
    /** Maximum size of objects that this proxy can handle. For the
     * moment set to 100 KB. You can adjust this as needed. */
    final static int MAX_OBJECT_SIZE = 10000000;
    String version;
    public int bytesRead;
    int status;
    String statusLine = "";
    String headers = "";
    byte[] body = new byte[MAX_OBJECT_SIZE];

    /** Read response from server. */
    @SuppressWarnings("deprecation")
        public HttpResponse(DataInputStream fromServer) {
```

```
/* Length of the object */
int length = -1;
boolean gotStatusLine = false;

/* First read status line and response headers */
try {
   String line = fromServer.readLine();
   while (line.length() != 0) {
                    if (!gotStatusLine) {
                       statusLine = line;
                       gotStatusLine = true;
                    } else {
                       headers += line + CRLF;
                    }
                    if (line.startsWith("Content-length") ||
                       line.startsWith("Content-Length")) {
                       String[] tmp = line.split(" ");
                       length = Integer.parseInt(tmp[1]);
                    }
                    line = fromServer.readLine();
   }
   System.out.println(statusLine);
} catch (IOException e) {
   System.out.println("Error reading headers from server: " + e);
   return;
}

try {
   bytesRead = 0;
   byte buf[] = new byte[BUF_SIZE];
   boolean loop = false;

   if (length == -1) {
          loop = true;
   }

   while (bytesRead < length || loop) {
          // Read it in as binary data
                    int res = fromServer.read(buf, 0, BUF_SIZE);
                    if (res == -1) {
                       break;
                    }
                    for (int i = 0; i < res && (i + bytesRead) < MAX_OBJECT_SIZE; i++) {
                       body[bytesRead+i] = buf[i];
                    }
                    bytesRead += res;
   }
```

```
                }
                catch (IOException e) {
                   System.out.println("Error reading response body: " + e);
                   return;
                }
        }

   //Checks if response status indicates modified
   public Boolean checkModified(){
        String[] tmp = statusLine.split(" ");
        if(tmp[1].equals("304")){
                return true;
        }
        return false;
   }

   //Return an array of headers used to check if modified
   public String[] getModified(){
        String[] modified;
        if(getEtag().equals("NULL")){
                modified = new String[]{
                        getLastModified()
        };
        }
        else{
                modified = new String[]{
                        getLastModified(),
                        getEtag()
        };
        }
        return modified;
   }

   //Gets the header Last-Modified
   public String getLastModified(){
        String[] indHeaders = headers.split(CRLF);
        for(int i = 0; i < indHeaders.length; i++){
                String[] tmp = indHeaders[i].split(" ");
                if(tmp[0].equals("Last-Modified:")){
                        String lastModified = "";
                        for(int j = 1; j < tmp.length; j++){
                                if(tmp[j].equals("GMT")){
                                        lastModified += tmp[j];
                                }
                                else{
                                        lastModified += tmp[j] + " ";
                                }
```

```
                }
                return lastModified;
            }
        }
        return "'Last-Modified' not found.";
    }

    //Gets the header ETag
    public String getEtag(){
        String[] indHeaders = headers.split(CRLF);
        for(int i = 0; i < indHeaders.length; i++){
            String[] tmp = indHeaders[i].split(" ");
            if(tmp[0].equals("ETag:")){
                String lastModified = tmp[1];
                return lastModified;
            }
        }
        return "NULL";
    }

    public String toString() {
        String res = "";

        res = statusLine + CRLF;
        res += headers;
        res += CRLF;

        return res;
    }
}
```