Tobias Hallen
16322468
26/02/2019

CS3031 Project 1

A Proxy Server

Proxy Servers:

A proxy server is a server which accepts requests for files from a client and forwards them to the required server on their behalf. The benefit of a proxy server is that it allows the proxy's administrators some modicum of control over the content the clients are permitted to gain access to. For this project, the local proxy was required to be able to dynamically block selected URLs, which could be expanded into phrase-blocking or quite easily.
It also allows for the implementation of a local cache, which benefits the client by offering much faster response times by saving oft-requested files locally on-disk. This allows the server to respond almost instantaneously to requests for cached files, as the proxy no longer needs to request the file from a server, which comparatively takes a long time.
This is especially true of the local proxy that was created for the sake of this project, but is also true of proxies which are simply hosted on the same network, though to a lesser degree.

Design:

The proxy had to be implemented to support multithreading, so as to be able to service multiple clients at the same time. Therefore, it consists of a "driver" class, which handles the Graphical User Interface, along with spawning threads to handle each incoming request, be it HTTP or HTTPS. Each new incoming request spawns a thread to service said request and client independently, which allows the server to handle several clients' incoming requests at the same time.
This "driver" class, HTTPProxy, is also responsible for the cache management system and maintaining the block list, which contains domain names blocked via the console gui. Both the block list and the cache list were implemented using hashmaps to facilitate quick lookup speeds, even when dealing with large volumes of file - and domain names. On startup, these two hashmaps are loaded with values from existing files if possible, and upon shutting down the server, they are saved onto said files in order to maintain block list and cache list integrity across several sessions.
Of course the console GUI allows users to list the current contents of both the cache and the block list, along with being able to dynamically add and remove domain names/phrases to the block list in-session. This functionality is not currently implemented for the cache, which leaves room for expansion in the future. The GUI does however allow users to list the current contents of the cache. This is implemented using the javax.swing library to create an intuitive, easy to use UI, featuring a scrolling window with all current and past connection info, and a panel to handle user IO such as blocking and unblocking, as well as relaying a "domain is blocked" message.
The other important class in this implementation is the worker thread class of the proxy server, HTTPProxyWorkerThread. This is the thread class which is spawned whenever the proxy server receives a new request, and handles the servicing of said request.

There are two main categories of request which the servicing thread may encounter - HTTP and HTTPS requests. The former are a simple task to service, discounting the added overhead of cache implementation.
Incoming requests are checked for whether they are HTTPS or not. This, parsing the requests, identifying file types and formatting them for cache storage, surprisingly proved one of the more difficult aspects of the project, as requests could have unusual syntax, and do not necessarily end in their respective file extension. Special reservations had to be made for image files, which had to be stored in a different format.
If the file is not contained in the cache, the thread will consult the cache list to determine whether it contains the requested resource. If the file is contained within the cache, the thread will open the file from the cache directory instead of consulting the server, and will echo the contents of the file directly to the client. This practice should provide a noticeable speed increase if certain files are being requested frequently. However, it does result in some small overhead as the searching of the cache does take some time.
If the file is not contained in the cache, the thread will contact the server as usual, echoing the client's request to the server, and echoing the server's response to the request back to the client. If possible, at this point, a local copy of the file is saved in the cache directory for future use.

More interesting, perhaps, are the HTTPS requests. When an HTTPS request comes in, the data transferred between client and server is encrypted.
However, the initial connection request made by the client is normal HTTP and can therefore be interpreted. This request contains the destination server the client would like to make the connection to, and is therefore used by the thread to establish a socket connection to said server, upon which a confirmation message is sent to the client. This process is known as CONNECT tunneling.
Any incoming data from the server is now forwarded directly to the client, and a listener thread is launched in order to forward any data that the client wishes to send to the server simultaneously.
The proxy server only handles the initial connection request data, and the rest is encrypted and cannot be interpreted. Therefore, the HTTPS data cannot be cached as HTTP requests can.
It is crucial to handle HTTPS requests on a proxy server, as the standard is becoming more and more widespread throughout the internet.

The final important class is ClientServerThread, mentioned above, which serves as a listener class for the client. It will read any data sent by the client into a buffer, and forward it to the server.

There is an additional supplemental utility class called MyPrintStream which serves to add the current formatted date to any message printed using System.out.println.

It is worth mentioning at this point that any exceptions caused by socket timeouts or faulty requests made while browsing are written directly to an exception.txt file which is flushed on startup, so as to only contain data from the last session. This allows users to see and interact with these exceptions, but prevents them from flooding the console.

Appendix:

Note: the Appendix Contains the code Listings, but for legibility reasons I will also include the source files in the submission

HTTPProxy:

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.ConcurrentModificationException;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.text.DefaultCaret;
```

```
public class HTTPProxy
{
	private static JFrame proxyGUI;

	//creates instance of HTTP proxy
	public static void main(String[] args) throws UnsupportedLookAndFeelException
	{
		UIManager.setLookAndFeel(UIManager.getLookAndFeel());
		HTTPProxy proxy = new HTTPProxy(8080);
		EventQueue.invokeLater(new Runnable()
		{
			public void run()
			{
				try
				{
					proxy.initialize();
					proxyGUI.setVisible(true);
				}
				catch (Exception e)
				{
					e.printStackTrace();
				}
			}
		});
		proxy.listen();
	}

	final JFileChooser fc = new JFileChooser();
	private JTextArea txtAbout;
	private JTextField iOField;

	private static JTextArea connectionInfoArea;
	private static JTextArea userIOArea;

	private static JScrollPane connInfoScrollPane;
	private static JScrollPane userIOScrollPane;


	private ServerSocket serverSocket;
	private volatile boolean running = true;
	static HashMap<String, File> cache;
	static HashMap<String, String> blockList;
	static ArrayList<Thread> threadList;

	FileWriter exceptionWriter;
	PrintWriter exceptionPW;
```

```
@SuppressWarnings("unchecked")
        public HTTPProxy(int port)
        {
                cache = new HashMap<>();
                blockList = new HashMap<>();
                threadList = new ArrayList<>();
                System.setOut(new MyPrintStream(System.out));
                try
                {
                        RandomAccessFile raf = new RandomAccessFile("exception.txt", "rw");
                        raf.setLength(0);
                        raf.close();
                        exceptionWriter=new FileWriter("exception.txt",true);
                        exceptionPW=new PrintWriter(exceptionWriter,true);
                        File cacheFile = new File("cacheList.txt");
                        if(!cacheFile.exists())
                        {
                                cacheFile.createNewFile();
                        }
                        else
                        {
                                FileInputStream fileStream = new
FileInputStream(cacheFile);
                                ObjectInputStream objectStream = new
ObjectInputStream(fileStream);
                                cache = (HashMap<String,File>)objectStream.readObject();
                                fileStream.close();
                                objectStream.close();
                        }
                        File blockFile = new File("blockedList.txt");
                        if(!blockFile.exists())
                        {
                                blockFile.createNewFile();
                        }
                        else
                        {
                                FileInputStream fileStream = new
FileInputStream(blockFile);
                                ObjectInputStream objectStream = new
ObjectInputStream(fileStream);
                                blockList = (HashMap<String,
String>)objectStream.readObject();
                                fileStream.close();
                                objectStream.close();
                        }
                }
                catch (IOException e)
                {
                        exceptionPW.write("Error loading previously cached sites file");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
```

```
                        e.printStackTrace(exceptionPW);
                }
                catch (ClassNotFoundException e)
                {
                        exceptionPW.write("Class not found loading in preivously cached
sites file");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }

                try
                {
                        serverSocket = new ServerSocket(port);
                        System.out.println("Proxy Started using port " +
serverSocket.getLocalPort() + ".");
                        running = true;
                }
                catch (SocketException e)
                {
                        exceptionPW.write("Socket Error");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }
                catch (SocketTimeoutException e)
                {
                        exceptionPW.write("Timeout Error\n");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }
                catch (IOException e)
                {
                        System.out.println("Read/Write Error");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }
        }
```

```
	public void listen()
	{
		while(running)
		{
			try {
				Socket s = serverSocket.accept();
				Thread t = new Thread(new HTTPProxyWorkerThread(s));

				threadList.add(t);
				t.start();
			}
			catch (SocketException e)
			{
				System.out.println("Server Shut Down...");
			}
			catch (IOException e)
			{
				exceptionPW.write(new Date().toString()); // Adding the
date
				exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
				e.printStackTrace(exceptionPW);
			}
			catch (NullPointerException e)
			{
				exceptionPW.write(new Date().toString()); // Adding the
date
				exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
				e.printStackTrace(exceptionPW);
			}
		}
	}

	public void shutDownProxy()
	{
		System.out.println("Shutting down Proxy..");
		running = false;
		try{
			FileOutputStream fileStream = new
FileOutputStream("cacheList.txt");
			ObjectOutputStream objectStream = new
ObjectOutputStream(fileStream);

			objectStream.writeObject(cache);
			System.out.println("Cashed Sites Saved");

			fileStream = new FileOutputStream("blockedList.txt");
			objectStream = new ObjectOutputStream(fileStream);
			objectStream.writeObject(blockList);
			objectStream.close();
			fileStream.close();
```

```
                        System.out.println("Blocked sites saved");
                        try
                        {
                                //go through and close all the threads
                                Iterator<Thread> i=threadList.iterator();
                                while(i.hasNext())
                                {
                                        if(i.next().isAlive())
                                        {
                                                i.next().join();
                                        }
                                }
                        }
                        catch (InterruptedException e)
                        {
                                exceptionPW.write(new Date().toString()); // Adding the
date
                                exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                                e.printStackTrace(exceptionPW);
                        }
                        catch (ConcurrentModificationException e)
                        {
                                exceptionPW.write(new Date().toString()); // Adding the
date
                                exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                                e.printStackTrace(exceptionPW);
                        }
                        catch (NoSuchElementException e)
                        {
                                exceptionPW.write(new Date().toString()); // Adding the
date
                                exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                                e.printStackTrace(exceptionPW);
                        }

                }
                catch (IOException e)
                {
                        exceptionPW.write("File read/write error");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }
                try
                {
                        serverSocket.close();
                }
                catch (Exception e)
                {
```

```
                        exceptionPW.write("Error closing socket");
                        exceptionPW.write(new Date().toString()); // Adding the date
                        exceptionPW.write(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date())+"\n"); // Formatted date
                        e.printStackTrace(exceptionPW);
                }
        }

        public static File searchCache(String url)
        {
                return cache.get(url);
        }

        public static void addToCache(String url, File file)
        {
                cache.put(url, file);
        }

        public static void addBlocked(String url)
        {
                blockList.put(url, url);
        }

        public static boolean blocked (String url)
        {
                if(blockList.containsKey(url)) return true;
                else return false;
        }

        public static boolean blockedPhrase (String url)
        {
                for(String s : blockList.values())
                {
                        if(url.contains(s))return true;
                }
                return false;
        }

        private void initialize()
        {
                proxyGUI = new JFrame();
                proxyGUI.addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent e)
                        {
                                proxyGUI.dispose();
                                shutDownProxy();
                        }
                });
                proxyGUI.setTitle("Proxy");
                proxyGUI.setBounds(100, 100, 913, 400);
                proxyGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                proxyGUI.setResizable(false);
```

```
		JMenuBar menuBar = new JMenuBar();
		proxyGUI.setJMenuBar(menuBar);

		JMenu mnHelp = new JMenu("Help");
		menuBar.add(mnHelp);

		JMenuItem mntmAbout = new JMenuItem("Usage");
		mntmAbout.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				JFrame frmAbout = new JFrame("How to Use");
				int X = proxyGUI.getWidth() / 2;
				int Y = proxyGUI.getHeight() / 2;
				frmAbout.setBounds(X, Y, 300, 150);
				frmAbout.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
				frmAbout.setResizable(false);
				txtAbout = new JTextArea();
				txtAbout.setText("First box is input, second box is
output.\nEnter a phrase and press corresponding \nbutton to add/remove from block
list.");
				frmAbout.getContentPane().add(txtAbout,
BorderLayout.CENTER);
				txtAbout.setEditable(false);
				frmAbout.pack();
				frmAbout.setVisible(true);
			}
		});
		mnHelp.add(mntmAbout);
		proxyGUI.getContentPane().setLayout(null);

		JButton blockPhrase = new JButton("Block Phrase");
		blockPhrase.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				String hostToBlock = iOField.getText();
				if (!hostToBlock.isEmpty())
				{
					iOField.setText("");
					int choice = JOptionPane.showConfirmDialog(null,"Are
you sure you want block: " + hostToBlock, "Block Phrase",JOptionPane.YES_NO_OPTION);
					if (choice == JOptionPane.YES_OPTION)
					{
						addBlocked(hostToBlock);
						addToInfoArea(hostToBlock + " has been
blocked.");
					}
				} else addToInfoArea("You must enter a phrase to block.");
			}
		});
		blockPhrase.setBounds(12, 12, 119, 100);
```

```
		proxyGUI.getContentPane().add(blockPhrase);

		JButton unblockPhrase = new JButton("Unblock Phrase");
		unblockPhrase.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				String phraseToUnblock = iOField.getText();
				iOField.setText("");
				int choice = JOptionPane.showConfirmDialog(null, "Are you
sure you want unblock: " + phraseToUnblock, "Unblock Phrase",
JOptionPane.YES_NO_OPTION);
				if (choice == JOptionPane.YES_OPTION)
				{
					if (blockList.remove(phraseToUnblock,
phraseToUnblock)) addToInfoArea("Unblocked: " + phraseToUnblock );
					else addToInfoArea("Cannot unblock: " +
phraseToUnblock + ", may not be present in list.");
				}
			}
		});
		unblockPhrase.setBounds(139, 12, 119, 100);
		proxyGUI.getContentPane().add(unblockPhrase);

		JButton listBlocked = new JButton("List Blocked");
		listBlocked.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				String temp = "";
				List<String> list = new
ArrayList<String>(blockList.values());
				for (int i = 0; i < list.size(); i++) {
					if (!list.get(i).isEmpty()) {
						temp += "[" + list.get(i) + "]";
					}
				}
				if (!temp.isEmpty()) {
					addToInfoArea("Blocked host: " + temp);
				} else {
					addToInfoArea("Block List is Empty.");
				}
			}
		});
		listBlocked.setBounds(266, 12, 119, 100);
		proxyGUI.getContentPane().add(listBlocked);

		JButton showCacheButton = new JButton("Show Cache");
		showCacheButton.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				addToInfoArea("Cache Dump:\n");
				for (String key : cache.keySet())
				{
```

```
                                addToInfoArea(""+key);
                        }
                        addToInfoArea("\n");
                }
        });
        showCacheButton.setBounds(393, 12, 119, 100);
        proxyGUI.getContentPane().add(showCacheButton);


        connectionInfoArea = new JTextArea(10, 40);
        userIOArea = new JTextArea(40,10);

        connectionInfoArea.setLineWrap(true);
        userIOArea.setLineWrap(true);

        connInfoScrollPane = new JScrollPane(connectionInfoArea);
        userIOScrollPane = new JScrollPane(userIOArea);

        connInfoScrollPane.setBounds(12, 122, 500, 190);
        userIOScrollPane.setBounds(513, 12, 380, 260);


connInfoScrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALW
AYS);

userIOScrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAY
S);

        proxyGUI.getContentPane().add(connInfoScrollPane);
        proxyGUI.getContentPane().add(userIOScrollPane);

        DefaultCaret connCaret = (DefaultCaret) connectionInfoArea.getCaret();
        connCaret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
        DefaultCaret iOCaret = (DefaultCaret) userIOArea.getCaret();
        iOCaret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);

        iOField = new JTextField();

        iOField.setBounds(513, 280, 380, 32);

        proxyGUI.getContentPane().add(iOField);

        iOField.setColumns(10);
    }

    public static void addToConnArea(String s)
    {
        s += "\n";
        try
        {
            connectionInfoArea.append(s);
        }
        catch (NullPointerException e) {}
```

```
        }

        public static void addToInfoArea(String s) {
                s += "\n";
                userIOArea.append(s);
        }
}
```

HTTPProxyWorkerThread:

```
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.imageio.ImageIO;

public class HTTPProxyWorkerThread implements Runnable
{

        FileWriter exceptionWriter;
        PrintWriter exceptionPW;
        private Thread httpsThread;
        Socket client;
        BufferedReader clientReader;
```

```
        BufferedWriter clientWriter;
        /**
         * Constructor for client request handler
         * @param socket socket connected to the client
         * READ
         */
        public HTTPProxyWorkerThread(Socket socket)
        {
                this.client = socket;
                try
                {
                        this.client.setSoTimeout(3000);
                        clientReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                        clientWriter = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                        exceptionWriter=new FileWriter("exception.txt",true);
                        exceptionPW=new PrintWriter(exceptionWriter,true);
                }
                catch (IOException e)
                {
                        exceptionPW.write("Read/Write Error\n");
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                }
        }



        /**
         * Parses Request, initialises appropriate response based on request type
         */
        @Override
        public void run()
        {

                //attempt to receive client request
                String request;
                try
                {
                        request = clientReader.readLine();
                }
                catch (IOException e)
                {
                        exceptionPW.write("Read/Write Error\n");
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                        return;
                }
```

```
                //format request into two strings containing the request and the URL
respectively
                String[] formattedArray = formatRequest(request);

                // check if requested site is in block list
                if(HTTPProxy.blockedPhrase(formattedArray[1]))
                {
                        HTTPProxy.addToInfoArea(formattedArray[1]+" has been blocked on
this proxy.");
                        return;
                }

                // Check if received HTTPS request
                if(formattedArray[0].equals("CONNECT"))
                {
                        HTTPProxy.addToConnArea("HTTPS Request Received: " +
formattedArray[1] + "\n");
                        httpsHandler(formattedArray[1]);
                }

                // Check if request is stashed in cache
                else
                {
                        File f;
                        if((f = HTTPProxy.searchCache(formattedArray[1])) != null)
                        {
                                HTTPProxy.addToConnArea("Found cached copy of: " +
formattedArray[1] + "\n");
                                getFromCache(f);
                        }
                        else
                        {
                                HTTPProxy.addToConnArea("No cached copy found, GET-ting: "
+ formattedArray[1] + "\n");
                                sendToClient(formattedArray[1]);
                        }
                }
        }

        /**
         * Deals with HTTPS connections/requests
         * @param url desired file to be transmitted over HTTPS
         */
        private void httpsHandler(String url)
        {
                //cut out the "http://" from the url
                String onlyURL = url.substring(7);

                //split the URL into actual URL and port
                String split[] = onlyURL.split(":");
                onlyURL = split[0];
                int port  = Integer.valueOf(split[1]);
```

```
                try
                {
                        //use DNS to get IP from URL
                        InetAddress ip = InetAddress.getByName(onlyURL);

                        //read in the HTTPS request
                        for(int i=0;i<5;i++)clientReader.readLine();

                        //create socket for server
                        Socket serverSocket = new Socket(ip, port);
                        serverSocket.setSoTimeout(4000);

                        //respond positively to client
                        String s = "HTTP/1.0 200 Connection established\r\n\r\n";
                        clientWriter.write(s);
                        clientWriter.flush();


                        //reader and writer to handle data forwarding between client and
server
                        BufferedWriter writeToServer = new BufferedWriter(new
OutputStreamWriter(serverSocket.getOutputStream()));
                        BufferedReader readFromServer = new BufferedReader(new
InputStreamReader(serverSocket.getInputStream()));

                        //open a new thread to handle data forwarding from client to server
                        ClientServerThread clientServerThread = new
ClientServerThread(client.getInputStream(), serverSocket.getOutputStream());

                        httpsThread = new Thread(clientServerThread);
                        httpsThread.start();

                        try
                        {
                                //read raw data from server to send it on to the client
                                byte[] buffer = new byte[4096];
                                int readTemp = serverSocket.getInputStream().read(buffer);
                                while(readTemp>=0)
                                {
                                        if (readTemp > 0)
                                        {
                                                client.getOutputStream().write(buffer, 0,
readTemp);
                                                if (serverSocket.getInputStream().available()
< 1)
                                                {
                                                        client.getOutputStream().flush();
                                                }
                                        }
                                        readTemp =
serverSocket.getInputStream().read(buffer);
                                }
                        }
```

```
                        catch (SocketTimeoutException e)
                        {
                                exceptionPW.write("HTTPS Time Out Error\n");
                                exceptionPW.write("\n"+new Date().toString()+"\n"); //
Adding the date

                                e.printStackTrace(exceptionPW);
                        }
                        catch (IOException e)
                        {
                                exceptionPW.write("Read/Write Error\n");
                                exceptionPW.write("\n"+new Date().toString()+"\n"); //
Adding the date

                                e.printStackTrace(exceptionPW);
                        }

                        try
                        {
                                serverSocket.close();
                                readFromServer.close();
                                writeToServer.close();
                                clientWriter.close();
                        } catch (NullPointerException e) {}

                }
                catch (SocketTimeoutException e)
                {
                        String line = "HTTP/1.0 504\n";
                        try
                        {
                                clientWriter.write(line);
                                clientWriter.flush();
                        }
                        catch (IOException e1)
                        {
                                exceptionPW.write("Read/Write Error\n");
                                e1.printStackTrace(exceptionPW);
                        }
                }
                catch (Exception e)
                {
                        exceptionPW.write("Error processing HTTPS request: " + url +"\n");
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                }
        }



        /**
```

```
         * Forwards file to client
         * @param url URL of requested file
         */
        private void sendToClient(String url)
        {
                try
                {
                        //transform URL into OS compatible filename while still being
identifiable
                        int fileExtensionIndex = url.lastIndexOf(".");
                        String extension;

                        //get the extension of the file
                        extension = url.substring(fileExtensionIndex, url.length());

                        //get everything but the extension of the file
                        String fileName = url.substring(0,fileExtensionIndex);


                        //get rid of sub domain
                        fileName = fileName.substring(fileName.indexOf('.')+1);

                        //replace any other illegal characters
                        fileName = fileName.replace(".", "dot");
                        fileName = fileName.replace("/","slash");

                        //remove any illegal characters in extension and add ".html" to
complete file name

                        if(extension.contains("/"))
                        {
                                extension = extension.replace(".", "dot");
                                extension = extension.replace("/","slash");
                                extension += ".html";
                        }
                        if(extension.contains(".png"))
                        {
                                extension = extension.replace("?", "questionmark");
                                extension = extension.replace(".png","");
                                extension += ".png";
                        }
                        if(extension.contains(".gif"))
                        {
                                extension = extension.replace("?", "questionmark");
                                extension = extension.replace(".gif","");
                                extension += ".gif";
                        }
                        if(extension.contains(".jpg"))
                        {
                                extension = extension.replace("?", "questionmark");
                                extension = extension.replace(".jpg","");
                                extension += ".jpg";
                        }
```

```
                        if(extension.contains(".jpeg"))
                        {
                                extension = extension.replace("?", "questionmark");
                                extension = extension.replace(".jpeg","");
                                extension += ".jpeg";
                        }

                        fileName = fileName + extension;
                        if(extension.contains(".png"))extension=".png";
                        if(extension.contains(".gif"))extension=".gif";
                        if(extension.contains(".jpg"))extension=".jpg";
                        if(extension.contains(".jpeg"))extension=".jpeg";

                        //try to cache file
                        boolean caching = true;
                        File cacheFile = null;
                        BufferedWriter writeToCache = null;

                        try
                        {
                                cacheFile = new File("Cache/" + fileName);
                                //create file if not already existing
                                if(!cacheFile.exists())cacheFile.createNewFile();
                                writeToCache = new BufferedWriter(new
FileWriter(cacheFile));
                        }
                        catch (IOException e)
                        {
                                caching = false;
                                exceptionPW.write("Read/Write Error\n");
                                exceptionPW.write("\n"+new Date().toString()+"\n"); //
Adding the date


                                e.printStackTrace(exceptionPW);
                        }
                        catch (NullPointerException e)
                        {
                                exceptionPW.write("NullPointerException when trying to open
File\n");
                        }

                        //check if file is in a conventional image format
                        if((extension.contains(".gif")) || extension.contains(".jpeg")
||extension.contains(".jpg") || extension.contains(".png"))
                        {
                                //get new BufferedImage from URL
                                URL remoteURL = new URL(url);
                                BufferedImage imageBuffer = ImageIO.read(remoteURL);

                                //check that an image was received
                                if(imageBuffer != null)
                                {
                                        //if yes, write it to disk
```

```
                                        ImageIO.write(imageBuffer, extension.substring(1),
cacheFile);

                                        //respond positively
                                        String temp = "HTTP/1.0 200 OK\n\r\n";
                                        clientWriter.write(temp);
                                        clientWriter.flush();

                                        //forward image file to client
                                        ImageIO.write(imageBuffer, extension.substring(1),
client.getOutputStream());

                                }
                                //otherwise nothing was received
                                else
                                {
                                        HTTPProxy.addToConnArea("404, image not found." +
fileName);
                                        String error = "HTTP/1.0 404 NOT FOUND\n"  + "\r\n";
                                        clientWriter.write(error);
                                        clientWriter.flush();
                                        return;
                                }
                        }

                        //should be text file otherwise
                        else
                        {
                                URL serverURL = new URL(url);
                                //connect to the server
                                HttpURLConnection connectionToServer =
(HttpURLConnection)serverURL.openConnection();
                                connectionToServer.setDoOutput(true);
                                connectionToServer.setUseCaches(false);
                                connectionToServer.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
                                connectionToServer.setRequestProperty("charset", "utf-8");


                                BufferedReader serverReader = new BufferedReader(new
InputStreamReader(connectionToServer.getInputStream()));


                                //respond positively to client
                                String temp = "HTTP/1.0 200 OK\n\r\n";
                                clientWriter.write(temp);


                                //keep reading lines until serverReader is empty
                                while((temp = serverReader.readLine()) != null)
                                {
                                        //write to file for cache, assuming file
initialisation was successful
```

```
                                        if(caching)    writeToCache.write(temp);

                                        //forward lines to the client
                                        clientWriter.write(temp);
                                }
                                //close serverReader
                                if(serverReader != null)serverReader.close();
                                clientWriter.flush();
                        }


                        if(caching)
                        {
                                //add data to cache in main class
                                writeToCache.flush();
                                HTTPProxy.addToCache(url, cacheFile);
                        }

                        //close writeToCache
                        if(writeToCache != null)writeToCache.close();

                        //close clientWriter
                        if(clientWriter != null)clientWriter.close();
                }
                catch (Exception e)
                {
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                }
        }



        /**
         * Gets Specified File from Cache
         * @param file The file to be retrieved from the cache
         */
        private void getFromCache(File file){
                try
                {
                        //gets the file extension to identify its type
                        String fileExtension =
file.getName().substring(file.getName().lastIndexOf('.'));

                        String proxyResponse;
                        //check if file is in a conventional image format
                        if((fileExtension.contains(".gif")) ||
fileExtension.contains(".jpeg") || fileExtension.contains(".jpg") ||
fileExtension.contains(".png"))
                        {
                                BufferedImage imageBuffer = ImageIO.read(file);
```

```
                            if(imageBuffer == null )
                            {
                                    //check if image is null, if yes response negative
                                    HTTPProxy.addToConnArea("NullPointer getting:
"+file.getName());
                                    proxyResponse = "HTTP/1.0 404 NOT FOUND \n"  +
"\r\n";
                                    clientWriter.write(proxyResponse);
                                    clientWriter.flush();
                            }
                            else
                            {
                                    //check if image is null, if not response positive
                                    proxyResponse = "HTTP/1.0 200 OK\n\r\n";
                                    clientWriter.write(proxyResponse);
                                    clientWriter.flush();
                                    ImageIO.write(imageBuffer,
fileExtension.substring(1), client.getOutputStream());
                            }
                    }

                    //should be text file otherwise
                    else
                    {
                            BufferedReader textBuffer = new BufferedReader(new
InputStreamReader(new FileInputStream(file)));
                            proxyResponse = "HTTP/1.0 200 OK\n\r\n";
                            clientWriter.write(proxyResponse);
                            clientWriter.flush();

                            String temp;
                            //keep writing out lines until textBuffer is empty
                            while((temp = textBuffer.readLine()) !=
null){clientWriter.write(temp);}
                            clientWriter.flush();

                            //close textBuffer
                            if(textBuffer != null)textBuffer.close();
                    }


                    //close clientWriter
                    if(clientWriter != null)clientWriter.close();

            }
            catch (IOException e)
            {
                    exceptionPW.write("Read/Write Error\n");
                    exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date
                    e.printStackTrace(exceptionPW);
            }
    }
```

```
        /**
         * Simple Class to format incoming requests and return an array containing the
URL and the request type
         * @param request The incoming request
         */
        String[] formatRequest(String request)
        {
                String[]a = {"",""};
                if(request!=null)
                {
                        HTTPProxy.addToConnArea("Received Request: " + request);
                        String actualRequest = request.substring(0,request.indexOf(' '));
                        String url = request.substring(request.indexOf(' ')+1);
                        url = url.substring(0, url.indexOf(' '));
                        if(!url.substring(0,4).equals("http"))url = "http://" + url;
                        a[0]=actualRequest; a[1]= url;
                }
                return a;
        }
}
```

ClientServerThread:

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.SocketTimeoutException;
import java.text.SimpleDateFormat;
import java.util.Date;

class ClientServerThread implements Runnable
{
        InputStream clientInput;
        OutputStream serverOutput;
        FileWriter exceptionWriter;
        PrintWriter exceptionPW;


        public ClientServerThread(InputStream clientInput, OutputStream serverOutput)
        {
                this.clientInput = clientInput;
                this.serverOutput = serverOutput;
                try
                {
                        exceptionWriter=new FileWriter("exception.txt",true);
                }
                catch (IOException e)
```

```
                {
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                }
                exceptionPW=new PrintWriter(exceptionWriter,true);
        }

        @Override
        public void run()
        {
                try
                {
                        //read raw data from client to forward it to the server
                        byte[] buffer = new byte[4096];
                        int readTemp = clientInput.read(buffer);
                        while (readTemp >= 0)
                        {
                                if (readTemp > 0)
                                {
                                        serverOutput.write(buffer, 0, readTemp);
                                        if (clientInput.available() < 1)
                                        {
                                                serverOutput.flush();
                                        }
                                }
                                readTemp = clientInput.read(buffer);
                        }
                }
                catch (SocketTimeoutException e)
                {
                        exceptionPW.write("Client HTTPS Timeout\n");
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e.printStackTrace(exceptionPW);
                }
                catch (IOException e1)
                {
                        exceptionPW.write("Read/Write Error\n");
                        exceptionPW.write("\n"+new Date().toString()+"\n"); // Adding the
date

                        e1.printStackTrace(exceptionPW);

                }
        }
}
```

MyPrintStream:

```java
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Date;

public class MyPrintStream extends PrintStream
{

	public MyPrintStream(OutputStream out)
	{
		super(out);
	}

	@Override
	public void println(String string)
	{
		Date date = new Date();
		super.println("[" + date.toString() + "] " + string);
	}
}
```