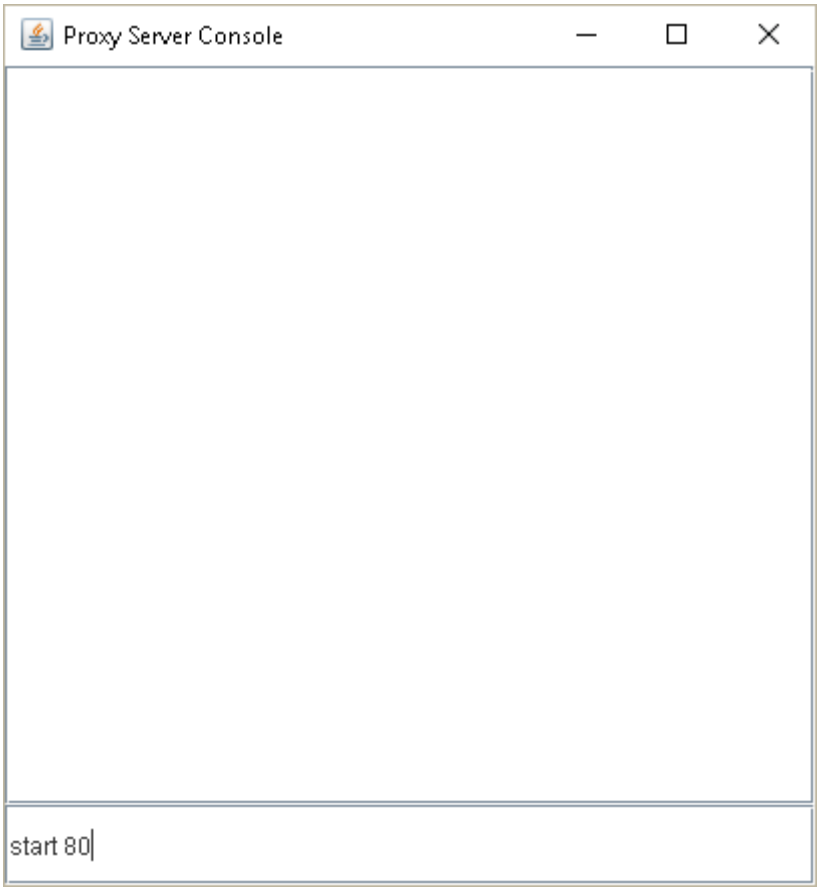
CS3031 Assignment 1 – Proxy Server

Yasir Mohamed – 13318246

22/02/16

Introduction

Included in my submission is a functioning Proxy Server for HTTP Requests. HTTPS requests do not work, however they are detected and outputted to the server. There is also a management console implemented which allows for dynamically blocking and unblocking domains, and the server implements caching. The server is run by compiling VisualConsole.java and running it. When the program begins, type “start (portnumber)” into the console to start the server, as can be seen below. Please note, if recompiling the program you must delete all existing class files in directory before recompiling.



Classes

VisualConsole.java

This class runs on the main thread of the server and is responsible for allowing the admin to interact with the Proxy Server. As such, I have called it the Management Console.

Figure 2.1 – The Proxy Server Console

There are a number of commands that can be run in the console, as detailed below:

start portNumber	Start the proxy server at a port specified by portNumber
Terminate	Terminate the proxy server
Block domain.com	Block clients from accessing domain.com
Unblock domain.com	Remove domain.com from the blocklist
Unblock all	Unblock all domains from the blocklist
Show blocklist	Show all domains currently blocked by the server

VisualBackend.java

This class is the backend interface of VisualConsole. Its only purpose is to parse the user input from the VisualConsole instance and perform some action if the inputted command was valid. If it wasn't, then an appropriate error message is returned.

ProxyServerSocket.java

This is the class that initialises the proxy server on a java serverSocket and accepts connections from clients. It is run on a separate thread from the management console. As this is the server thread, it also manages the block list. It has three main Server functions; startProxyServer ,run and stopServer.

startProxyServer initialises the server on a server socket and all its dependencies. It then calls run(), where it begins listening for a client to connect. When a client connects to the server, a new thread is spawned in the ClientConnectionHandler class to handle the connection, and the server listens for more connections. The StopServer method shuts down the proxy server.

ProxyServerSocket also handles the blocklist of domains that cannot be accessed on the server. Its functionality includes:

checkBlockList(String domain)	Check the blocklist for domain
blockDomain(String domain)	Add the specified domain to the blocklist
unblockDomain(String domain)	Remove the specified domain from the blocklist
dumpBlockList()	Unblock all the domains on the blocklist
getBlockList()	Return all the domains on the blocklist

ClientConnectionHandler.java

The ClientConnectionHandler class runs on a separate thread for each resource requested by the client. It acts as the middleman between the HttpRequest class and the ProxyServerSocket class.

When a ClientConnectionHandler is spawned, it reads in the request from the client and passes it into a HttpRequest object. The HttpRequest object then parses each line of content from the client and properly formats it to be sent by the server as a HTTP Request.

HttpRequest.java

This is my own custom class for handling HTTP request objects. Its function in the server is to parse data sent by the client and pack it into a HTTP Request that can be sent by the server, then relay the response from the destination server back to the client. It also displays details of requests made by clients. Its method summary can be seen below:

isRootFile()	Used In caching, this checks if the destination is a root file like google.com
checkMinRequirements()	A HTTP Request cannot be sent without having the minimum headers
printRequestDetails()	Print details of the request made by the client to the terminal
setHeaders()	Set the HTTP request headers
sendSecureRequest()	Send a HTTPS request. Doesn't actually work
sendUnsecureRequest()	Send a HTTP request to the destination
serveDenied()	If client tries to access a blocked domain, serve file detailing denial of access
respondToClient()	Send the requested resource back to the client
serveCachedFile()	If a cached version of the site is available, serve the cached version
processRequest()	When a HTTP request meets minimum requirements, begin process of sending the request
parseHeader()	Process the data passed in from the client

ServerCache.java

This class is responsible for implementing the caching functionality of the server. At its current implementation, this class saves the static html files of websites to a cache folder, but is unable to save dynamic page structures. This saves the index file of every website visited by clients on the server, but does not implement a replacement policy due to time constraints. As a result, there is no limit to the size of the cache and the server does not check if its cached file is outdated. When a client requests the index of a website and the index is not saved in cache, the server saves the file to the cache and then returns the cached version to the client. If the client requests a destination already indexed, the indexed file is returned to the client. If the client requests a resource that is not the index of a server, the resource is fetched and sent to the client.

The method summary of ServerCache.java can be seen below:

checkCache(String siteName)	Check if a site is already cached on the server
cacheSite(String siteName, String destination)	Cache the site index, as indicated by destination, to the server.