A Web Proxy Server

20 FEBRUARY
RUTH BRENNAN

ADBVANCED TELECOMMUNICATIONS
CSU34031

INSTRUCTIONS

The objective of the exercise is to implement a Web Proxy Server.
A Web proxy is a local server, which fetches items from the Web on behalf of a Web client instead of the client fetching them directly. This allows for caching of pages and access control.

The program should be able to:
1. Respond to HTTP & HTTPS requests, and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.
2. Handle Websocket connections.
3. Dynamically block selected URLs via the management console.
4. Efficiently cache requests locally and thus save bandwidth. You must gather timing and bandwidth data to prove the efficiency of your proxy.
5. Handle multiple requests simultaneously by implementing a threaded server.

The program can be written in a programming language of your choice. However, you must ensure that you do not overuse any API or Library functionality that implements the majority of the work for you.

DESIGN

HTTP & HTTPS

I implemented this web proxy in Node js using it's HTTP library.
Here I create a server object and the program to handle the requests is in the server class.

```
//SERVER STUFF
//create a server object:
var server = http.createServer(onRequest).listen(port,() => {
    console.log(`Server running at http://${hostname}:${port}/`);
  }); //the server object listens on port

function onRequest(req,res) {

    handle_server.onRequest(req,res,blockedurls,serverCache);
}
```

The on request function is part of the server file and it saperately makes the HTTp and HTTPS get requests using the http and https libraries. The handle request function then takes care of the response to that get request.
After checking that the status code is OK (200), it sets the standard encoding and handles the data received in the response before forwarding it to the brwoser.

Websocket

I implemented this web proxy in Node js using it's ws library.
Here I create a websocket server and the program to handle the requests is in the wss class (WebSocket Server).

```
//WEBSOCKET STUFF

// WebSocket server
var wss = new ws.Server({ server });

// Handle connections to WebSocket server
wss.on('connection', function connection(ws) {

  console.log("Received websocket connection...");

  ws.on('message', function incoming(message) {
    console.log('Received WebSocket request for: %s', message);

    handle_wss.handleWebSocketRequest(message, ws, blockedurls, serverCache);
  });

});
```

The on request function deals with the server requests but deals with it as a response to the client socket. The response to the get request is again handled in the handle request. It again checks the status code is correct, sets the encoding and the handles the response data so that it can be displayed.
The Websocket functionality works as the client file works as the client and the index file has the server functionality. A listener is set up to allow the client to request a url for example:

Request www.google.com

The program also gives the client the ability to block and unblock urls, this is explained below.

Blocking and unblocking URLS

The blocked urls are stored in a hastable and before a request is handled the hashtable is consulted. To block and unblock the url key is added or removed from the table. The management console is again implemented using a listener. The commands that are complient are block and unblock followed by the url.

Block www.google.com
Unblock www.google.com

```
// Listens to the console
//If block followed by url is entered -> Will block URL here
//If unlock followd by URL is entered -> will unblock URL here
//Also handles error cases such as incorrect command (not block/unblock) and deals with unblocking non blocked URLs
stdin.addListener("data", (data) => {

    //Get command {block, unblock}
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    if ( command == "block" ) {
        var url = data.toString().substring(6).trim();
        blockedurls.put(url);
        console.log("URL: " + url + " has been sucssfully blocked");

    } else if ( command == "unblock" ) {
        var url = data.toString().substring(8).trim();

        if( blockedurls.containsKey(url) ){
          blockedurls.remove(url);
          console.log("URL: " + url + " has been sucssfully unblocked")
        } else {
          console.log("URL: " + url + " is not blocked");
        }
    } else {
        console.log("Unknown command");
    }

});
```

Cache Requests

The cache is implemented using the cache library from Node js. I created the Cache items so that they expire after 10 minutes and the url acts as the identiifing key.

```
//CACHE
//stdTTL: (default: 0) the standard ttl as number in seconds for every generated cache element. 0 = unlimited
//checkperiod: (default: 600) The period in seconds, as a number, used for the automatic delete check interval. 0 = no periodic che
var serverCache = new cache({ stdTTL: 600, deleteOnExpire: true });
```

CODE

Index.js

```
/*
   What is a proxy server?
   Proxies act as intermediaries between clients and servers, they can preform processing or just forward requests upstream.
*/
/*
   What is a HTTP request?
   HTTP works as a request-response protocol between a client and server
*/
/*
   HTTP vs HTTPS?
   HTTPS is HTTP with encryption. The only difference between the two protocols is that HTTPS uses TLS (SSL) to
   encrypt normal HTTP requests and responses.
   As a result, HTTPS is far more secure than HTTP.
*/


//Load HTTP module
const http = require("http");
const https = require('https');
const hostname = 'localhost';
const port = 8000;
const SimpleHashTable = require('simple-hashtable');
const stdin = process.openStdin();
var cache = require( "node-cache" );
const { PerformanceObserver, performance } = require('perf_hooks');


const handle_wss = require('./wss.js');
const handle_server = require('./server.js');

//ws is a popular WebSockets library for Node. js. We'll use it to build a WebSockets server.
//It can also be used to implement a client, and use WebSockets to communicate between two backend services.
var ws = require('ws');


//CACHE
//stdTTL: (default: 0) the standard ttl as number in seconds for every generated cache element. 0 = unlimited
//checkperiod: (default: 600) The period in seconds, as a number, used for the automatic delete check interval. 0 = no periodic check.
var serverCache = new cache({ stdTTL: 600, deleteOnExpire: true });
```

```
//Create a hastable to store blocked URLs for ease of searching
var blockedurls = new SimpleHashTable();
// To have a site blocked at the beginning
blockedurls.put('www.google.com', 'blocked');

//SERVER STUFF
//create a server object:
var server = http.createServer(onRequest).listen(port,() => {
   console.log(`Server running at http://${hostname}:${port}/`);
 }); //the server object listens on port

function onRequest(req,res) {

   handle_server.onRequest(req,res,blockedurls,serverCache);
}


//WEBSOCKET STUFF

// WebSocket server
var wss = new ws.Server({ server });

// Handle connections to WebSocket server
wss.on('connection', function connection(ws) {

 console.log("Received websocket connection...");

 ws.on('message', function incoming(message) {
   console.log('Received WebSocket request for: %s', message);

   handle_wss.handleWebSocketRequest(message, ws, blockedurls, serverCache);
 });

});

//MANAGEMENT CONSOLE STUFF

// Listens to the console
//If block followed by url is entered -> Will block URL here
//If unlock followd by URL is entered -> will unblock URL here
//Also handles error cases such as incorrect command (not block/unblock) and deals with unblocking non blocked URLs
stdin.addListener("data", (data) => {
```

```
    //Get command {block, unblock}
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    if ( command == "block" ) {
        var url = data.toString().substring(6).trim();
        blockedurls.put(url);
        console.log("URL: " + url + " has been sucssfully blocked");

    } else if ( command == "unblock" ) {
        var url = data.toString().substring(8).trim();

        if( blockedurls.containsKey(url) ){
          blockedurls.remove(url);
          console.log("URL: " + url + " has been sucssfully unblocked")
        } else {
          console.log("URL: " + url + " is not blocked");
        }
    } else {
        console.log("Unknown command");
    }

});
```

Server.js

```
//Server class

//Load HTTP module
const http = require("http");
const https = require('https');
const url = require('url');
const SimpleHashTable = require('simple-hashtable');
var cache = require( "node-cache" );
const { PerformanceObserver, performance } = require('perf_hooks');

//Handle Server Response
function handleResponse(url, handle_res, res, blockedurls,serverCache){

    // Check if URL is blocked
    if ( blockedurls.containsKey(url) ) {
```

```
    console.log("URL " + url + " is blocked.");
    res.write("URL " + url + " is blocked.");
    res.end();
    return;
}

// Status code from the get response
const { statusCode } = handle_res;
var error;

//Check status code 200 means that all is OK otherwise handle response error
if ( statusCode !== 200 ) {
  error = new Error('Request Failed.\n' + `Status Code: ${statusCode}`);
  console.error(error.message);
  res.write(error.message);
  res.end();
  return;
}

var start = performance.now();
console.log("start "+start);

var hit = false;
// Check cache for web page and verify expires
try {
cacheRes = serverCache.get(url)
    if (cacheRes == undefined) {
        console.log("URL not in Cache");
    } else {
        res.write(cacheRes.body);
        res.end();
        console.log("Valid Cache");
        hit = true;
        var validEnd = performance.now();
        console.log("valid end " + validEnd);
    }
} catch {
    console.log("request cache get error");
}

if (!hit) {
    //encoding
    handle_res.setEncoding('utf8');
```

```
        var rawData = '';

        //At the stage when data is recieved
        handle_res.on('data', (chunk) => {
            rawData += chunk;
        });



        //At the end of the response
        handle_res.on('end', () => {

            //Make cache object to store in cache
            cacheObject = {
                body: rawData
            }

            //store cache object in the cache
            serverCache.set(url, cacheObject, (err, success) => {
                if (!err && success) {
                    console.log("URL added to cache");
                } else {
                    console.log("URL not added to cache");
                }
            });

            res.write(rawData);
            res.end();
            var missEnd = performance.now();
            console.log("miss end " + missEnd);
        });
    }

}


module.exports = {
    onRequest: function(req, res, blockedurls,serverCache) {

            var split = url.parse(req.url.substring(1), false);
            if (split.protocol!=null) {
                //res.write(split.protocol);  //  https:
                //res.write(split.path);      //  /
                //res.write(split.hostname);  //  www.tcd.ie
```

```
            //res.write(split.href);    //  https://www.tcd.ie/
            //res.end(); //end the response

            // Handle http and https request seperately
            console.log('\nReceived request for: ' + split.protocol + '//'+ split.hostname);
            if( split.protocol == 'http:' ) {

                http.get(split.href, (response) => handleResponse(split.hostname, response, res, blockedurls,serverCache))
                .on('error', (e) => {
                    console.error(`Got error: ${e.message}`);
                })
                .on('timeout', () => {
                    console.log('Request timeout');
                    res.end();
                    request.abort();
                });

            } else if (split.protocol == 'https:') {

                https.get(split.href, (response) => handleResponse(split.hostname,response, res,blockedurls,serverCache))
                .on('error', (e) => {
                    console.error(`error: ${e.message}`);
                })
                .on('timeout', () => {
                    console.log('Request timeout');
                    res.end();
                    request.abort();
                });

            } else {

                res.write('Invalid request');
                res.end();

            }
        }

    }
}
```

wss.js

```
//WebSocket Server (WSS) class
/*
   What is a WebSocket?
   A WebSocket is a persistent connection between a client and server.
   WebSockets provide a bidirectional, full-duplex communications channel that operates over HTTP through a single TCP/IP
   socket connection.
   At its core, the WebSocket protocol facilitates message passing between a client and server
*/

const http = require("http");
const https = require('https');
const url = require('url');
const SimpleHashTable = require('simple-hashtable');
const { PerformanceObserver, performance } = require('perf_hooks');

// Handle WebSocket responses
function handleWebSocketResponse(url, res, ws, blockedurls,serverCache) {

   // Check if URL is blocked
   if(blockedurls.containsKey(url)){
      console.log("URL " + url + " is blocked.");
      ws.send("URL " + url + " is blocked.");
      return;
   }

   // Status code from the get response
   const { statusCode } = res;

   var error;

   //Check status code 200 means that all is OK otherwise handle response error
   if ( statusCode !== 200 ) {
      error = new Error('Request Failed.\n' + `Status Code: ${statusCode}`);
      console.error(error.message);
      ws.send(error.message);
      return;
   }

   var hit = false;
   // Check cache for web page and verify expires
   try {
      cacheRes = serverCache.get(url)
```

```
        if (cacheRes == undefined) {
            console.log("URL not in Cache");
        } else {
            ws.send(cacheRes.body);
            console.log("Valid Cache");
            hit = true;
        }
    } catch {
        console.log("request cache get error");
    }

    if (!hit) {
        //encoding
        res.setEncoding('utf8');
        var rawData = '';

        //At the stage when data is recieved
        res.on('data', (chunk) => {
            rawData += chunk;
        });

        //At the end of the response
        res.on('end', () => {

            //Make cache object to store in cache
            cacheObject = {
                body: rawData
            }

            //store cache object in the cache
            serverCache.set(url, cacheObject, (err, success) => {
                if (!err && success) {
                    console.log("URL added to cache");
                } else {
                    console.log("URL not added to cache");
                }
            });

            ws.send(rawData);
            console.log("success");
        });
    }
}
```

```
module.exports = {

  // Handle WebSocket requests
  handleWebSocketRequest: function(URL, ws, blockedurls, serverCache) {
    var split = url.parse(URL, true);

    // Handle http and https request seperately
    console.log('\nReceived request for: ' + split.protocol + '//'+ split.hostname);
    if( split.protocol == 'http:' ) {

      http.get(split.href, (response) => handleWebSocketResponse(split.hostname, response, ws, blockedurls, serverCache))
      .on('error', (e) => {
        console.error(`Got error: ${e.message}`);
      })
      .on('timeout', () => {
        console.log('Request timeout');
        res.end();
        request.abort();
      });
    } else if (split.protocol == 'https:') {

      https.get(split.href, (response) => handleWebSocketResponse(split.hostname, response, ws, blockedurls, serverCache))
      .on('error', (e) => {
        console.error(`Got error: ${e.message}`);
      })
      .on('timeout', () => {
        console.log('Request timeout');
        res.end();
        request.abort();
      });

    } else {
      ws.send('Invalid request');
    }
  }
}
```

client.js

```
const WebSocket = require('ws');
const hostname = 'localhost';
const port = 8000;
var stdin = process.openStdin();
var validUrl = require('valid-url');

// Console input listener
stdin.addListener("data", function(data) {

    // Listens to the console
    //Enter request followed by url
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    if (command == "request") {

      var url = data.toString().substring(8).trim();
      if(validUrl.isUri(url)){
        console.log("URL OK, sending proxy request");
        ws.send(url);
      } else {
        console.log("Invalid URL");
      }

    } else {

      console.log("Unknown command");

    }
});

//  ws://localhost:8000
const ws = new WebSocket('ws://' + hostname + ':' + port);

ws.on('open', function open() {
  console.log("Successful connection to proxy");
});

ws.on('message', function incoming(mes) {
    console.log('Received response from proxy: %s', mes);
});
```

```
ws.on('close', function closed() {
  console.log("Proxy connection closed");
})
```

The above code base is also availible here in the github repo.