ITCS 6166/8166 – Computer Communications and Networks
Report – Project 01

HTTP Client and Server

Authors:

Stephen Stroud – UNCC ID:

Taylor Conners – UNCC ID: 800228848

Our project was constructed using C# with Microsoft's .NET CORE, and it was designed and tested in Microsoft's Visual Studio 2017 Community Edition. It consists of two components: an HTTP server that listens for and fulfills several requests using an HttpListener object, and an HTTP client that issues requests using an HttpClient object. Both the HttpListener and HttpClient classes are included in the .NET CORE architecture and work within the TCP protocol.

The server can be run through a standard terminal by issuing the following command: "dotnet run port_number" where port_number is the port with which the HttpListener will listen through and handle incoming HTTP requests. The server can handle two types of standard requests: a GET request, and a PUT request. When a GET request is issued to the server, it checks to see if the file requested currently exists in its local directory. If the file does currently exist, the server constructs a response with a status code of 200, signifying that the request was received and that it was received for a valid file, before returning the requested file to the HTTP client. If the file does not currently exist on the server, the server constructs a response with a status code of 404, signifying that the requested file could not be found on the server, and sends its response to the client.

The server handles a PUT requests by writing files to a local directory with a maximum buffer size of 2048 kilobytes. This number was decided upon arbitrarily and can be increased if it needs to be, but the size seemed appropriate for testing the PUT response, so we left it at that. When a file is successfully received by the server, it is written to a local directory (the same directory from which GET commands are responded to) with its original name and content intact. This was tested with .txt and .html files, and both appeared uncorrupted by the transfer for every test issued. The server's response is also issued with a status code of 200, if the response successfully writes the file to the server.

The greatest struggle that we shared was ensuring that a file was written to the server without corruption. Initially, we did this by reading the input stream as a string, and wrote all data to a generic file. This was not a sufficient solution, however, because it involved manually editing file extensions after transmission had been completed. We discovered, however, that it was more prudent to read the request's input stream using a BinaryReader, writing the input stream to a Byte array. We are satisfied with our solution.

References:

- https://www.youtube.com/watch?v=8d5JWDuG2Tg
- https://stackoverflow.com/questions/7004616/how-to-use-httplistener-to-receive-http-post-which-contain-xml
- https://www.techcoil.com/blog/the-http-response-and-how-it-relates-to-system-net-httplistener/
- https://stackoverflow.com/questions/381508/can-a-byte-array-be-written-to-a-file-in-c
- https://stackoverflow.com/questions/2934295/c-sharp-save-a-file-from-a-http-request
- https://stackoverflow.com/questions/9963052/sending-a-file-using-tcpclient-and-networkstream-in-c-sharp
- https://stackoverflow.com/questions/10182751/server-client-send-receive-simple-text
- https://social.msdn.microsoft.com/Forums/en-US/5fb9863e-9cf2-401e-a49e-2818c4908012/http-post-using-tcpclient?forum=netfxnetcom
- https://www.codeproject.com/Articles/32633/Sending-Files-using-TCP
- https://social.msdn.microsoft.com/Forums/en-US/27ee8b9d-3d86-4fa8-b311-f7fcaa5d419f/keep-tcplistener-listening-without-using-while-true-loop?forum=netfxnetcom