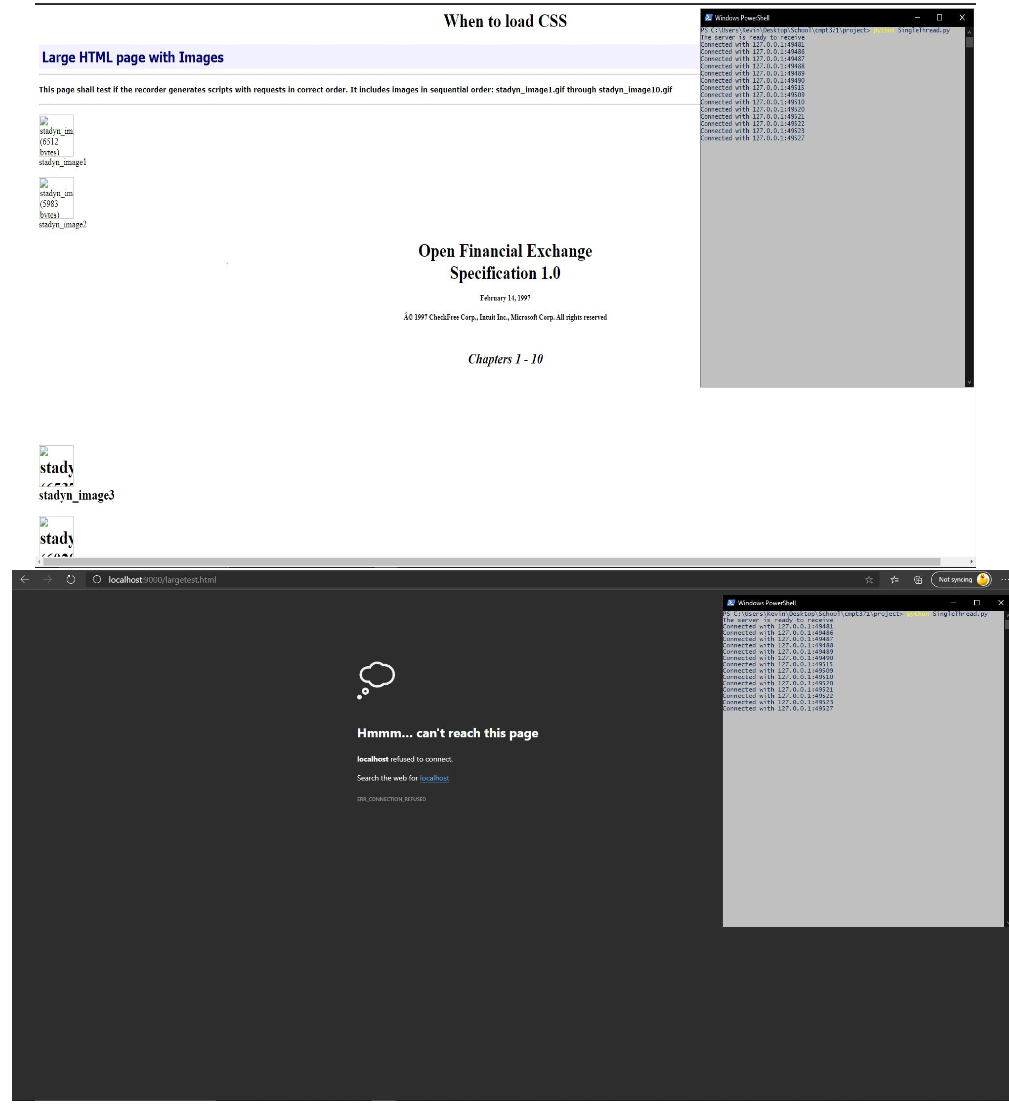
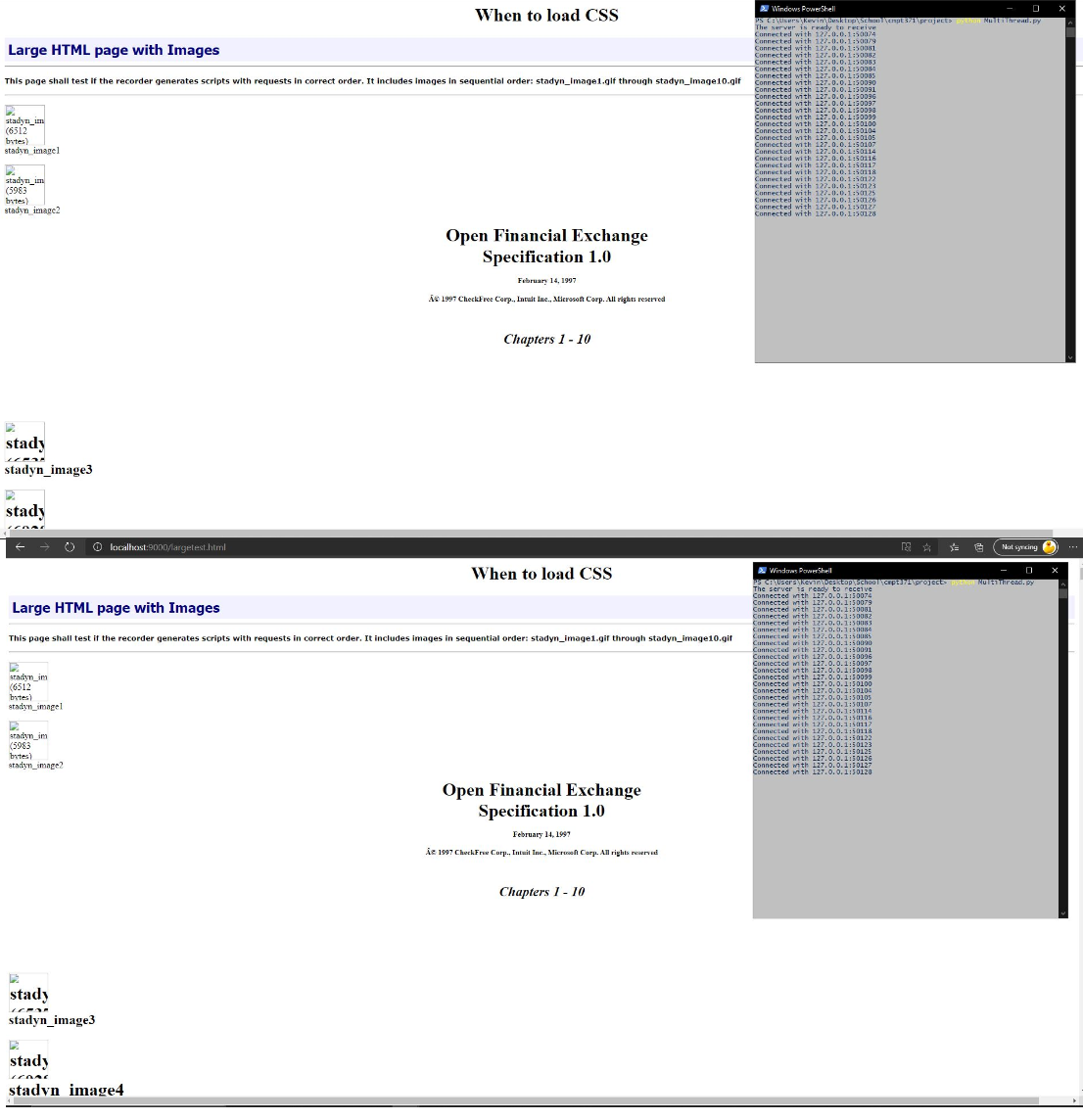
Step 5.

In order to see any multi-threaded benefits some changes had to be made. The given test file was too small and can be sent quick enough that it works under a single threaded process as well. A larger test file was made to slow down the send process. The socket.sendall also sent the large test file fast enough that single threaded worked so a slower send method was used. Instead of sending the whole message encoded, each character was encoded and sent one at a time. The testing involved sending a request from two browsers simultaneously with the address <http://localhost:9000/largetest.html>. The resulting outputs were:

Single-Threaded



Multi-Threaded

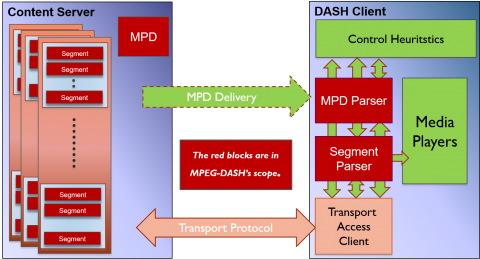


With single-threading only one instance was able to connect whereas with multithreading both instances were able to connect simultaneously.

Step 6

1) The server stores the content which in this case are the segmented video files with varying bitrates and resolutions. The server will listen to requests and when one is received it can send a video in chunks which are served constantly so that the client device can start displaying the data without having to download the entire video file before playing it.

2. The client side sends a request for a video file. The client monitors the network bandwidth and decides on how to adapt to the available bandwidth by fetching segments of different options. The client then receives the segments where a segment parser handles the segments and encapsulates them for the media player. The control heuristics and MPD parser determine the request options, interprets the mpd and controls how the segment parser should work for based on what is requested.

An example of the process: 

https://mpeg.chiariglione.org/news/dash-behind-scenes