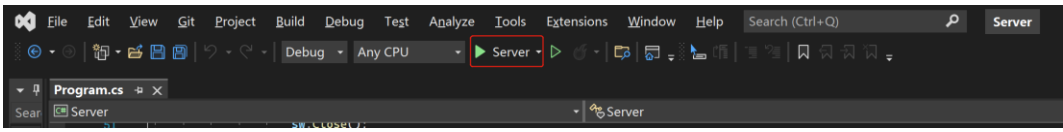
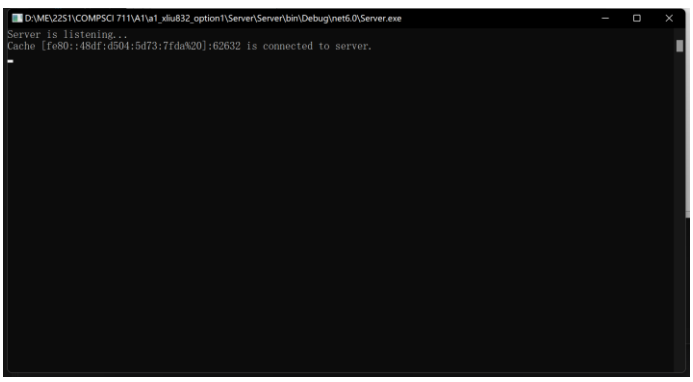
a) I implemented option 1.
b) I did not test the program on a lab machine.
c) To run the program, there are three ways:
    a. Method 1:
        i. Double click the batch file to run the three programs (at \a1_xliu832_option1\compileC#.bat). It opens up Server.exe, Cache.exe, and Client.exe. If the batch file fails to execute the three programs, please use method 2 or method 3.
    b. Method 2:
        i. Inside the Server folder (\a1_xliu832_option1\Server), open the Server.sln file using Visual Studio 2022.
        ii. Inside the Cache folder (\a1_xliu832_option1\Cache), open the Cache.sln file using Visual Studio 2022.
        iii. Inside the Client folder (\a1_xliu832_option1\Client), open the Client.sln file using Visual Studio 2022.
        iv. Inside the Visual Studio 2022 window, click on the green arrow to run the program. **Note**: please start the three programs in this order (Server → Cache → Client). If the three programs are opened in a wrong order, the connection between them will fail.

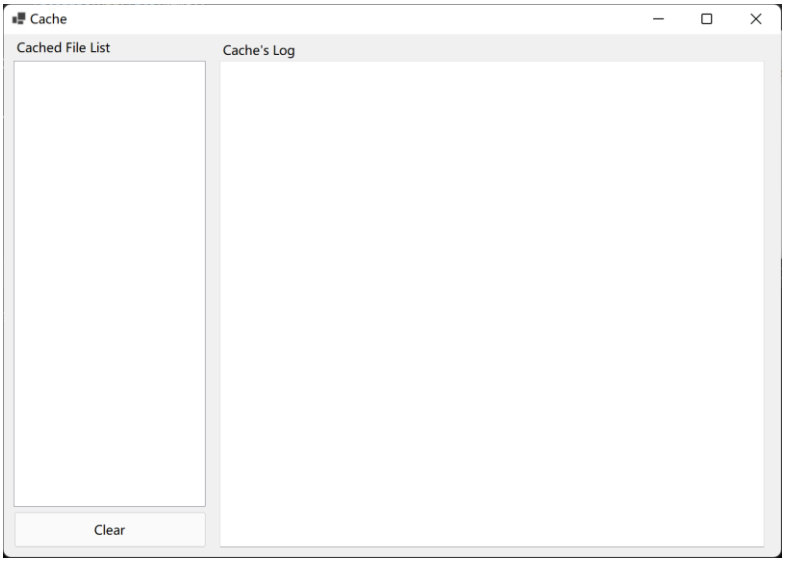


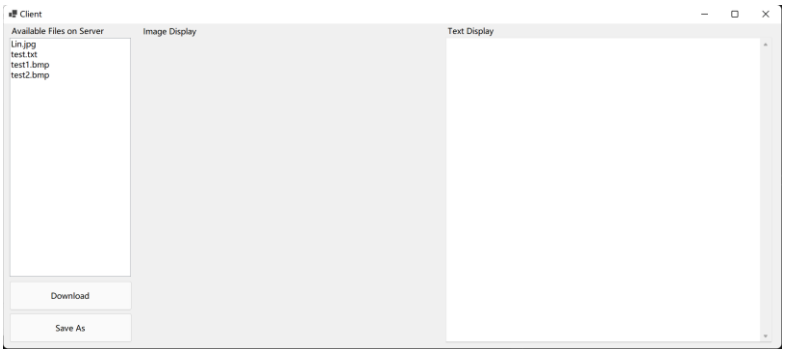
    c. Method 3:
        i. Go to \a1_xliu832_option1\Server\Server\bin\Debug\net6.0, double click on Server.exe
        ii. Go to \a1_xliu832_option1\Cache\Cache\bin\Debug\net6.0-windows, double click on Cache.exe
        iii. Go to \a1_xliu832_option1\Client\Client\bin\Debug\net6.0-windows, double click on Client.exe

The three windows will pop up.



If the server is successfully built, it will display “Server is listening…”





The test images and txt files are stored in \a1_xliu832_option1\Images. Select the file you want to retrieve in the list on the left, click the Download button to get the file from server. The requested image file will display in the middle area, the requested text file will display in the right area. You can click on the Save As button to save the requested file. The files will be saved in \a1_xliu832_option1\download.

d) In option 1, we only need to keep a Dictionary object in cache to keep a record of files requested by client. Every time when client requests a file that has previously been sent from server to client, the cache will send the stored file to client straightaway, so that the request and response do not need to go to the server again. This reduces the response time and less information will be transferred for a given bandwidth comparing to communicating between machines without cache.
In option 2, the implementation value-based web caching is required. We need to partition a file into fragments, and keep record of the fragments in the cache and server. Every time when client requests a file, the cache will check if it has the copies of fragment that the requested file contains. If cache has data fragments that matched part of the requested file, then it will send request to the server to download the remaining data fragments. In this way, the response latency is even

lower than the implementation in option 1, because the amount of data transferred between the three machines is decreased (option 1 transfers a whole file of data, option 2 only transfers fragments of data) and so less bandwidth will be used.
A message digest function or a Rabin function are possible implementations of value-based web caching. They all partition the file into data fragments. These functions are uniform and random. However, the amount of computation carried out by the cache and the server will be larger, because they need to compute the partitioning of the files, and the cache needs to check through the data fragments to check which ones need to be retrieved from the server.