I implemented **option 1** and **have tested my program on a machine in the lab**.
About program running:
1. Use the batch process **startrun.bat** to run. After running, the clientGUI and serverGUI will pop up. First, click the "server" button of the server GUI to open the server. After the server is opened, perform any subsequent operations, list the file name, download the file content, the administrator from Select a file in the server's file system and place the selected file in the service client's directory. Use the "Open cache" button to open the cacheGUI, you can view the log and cache list and allow the emptying operation.
2. To run without batch processing, you need to run server, cache, and client in sequence, and then perform the above operations.
There is still a bug in the program, that is, when the user selects the last file, the program will pop up a path error, no matter what the last file is, the error will pop up, and it can work normally when the file is not the last one.

**Compare and contrast:**
Option 1 and Option 2 are two different caching strategies. In Option 1, the caching technique involves storing and retrieving entire files. Option 2 involves caching file fragments, not entire files.
**Response time:** In Option 1, the user-perceived response time will be faster for subsequent requests for the same file because the cached copy is available locally, while in Option 2 the user-perceived response time will be faster for the file that is available in the cache part of , but slower for the part that needs to be fetched from the server. Subsequent requests for the same file will be faster in option 2 if more fragments are already cached.
**Amount of network bandwidth:** The Option 1 technique applies to text files, which allows caches to serve subsequent requests more quickly, saving network bandwidth by reducing the number of requests that need to be sent to the server. Option 2 technology works with image files and allows for more efficient use of caching, which can save network bandwidth by downloading only missing fragments, and reduce computation performed by the origin server by not downloading the entire file.
**Calculation:** In Option 1, the cache needs to check the file system to see if the file has been cached, and if so, return the cached copy; otherwise, it downloads the file from the server. In Option 2, the cache needs to check which fragments of the file are already cached, and only download the missing fragments from the server, and then combine them to construct the requested file.
To sum up, in terms of user-perceived response time, Option 2 may have a slightly slower initial response time because it may need to download more pieces than Option 1 may need to download the entire file. However, subsequent requests for the same file may be faster in Option 2 if more fragments are already cached. The amount of network bandwidth that can be saved is likely to be higher in Option 2 because only the missing pieces are downloaded. Option 2 requires more computation as the cache server needs to fragment the files. This can lead to higher CPU usage and memory usage on the cache server. Option 1 is suitable for small files and frequent cache hits. On the other hand, Option 2 is suitable for large files, and only some files are frequently requested, or the file similarity is high, such as image files with only a few pixels different.