Char Design Document

Author: Brendan Teo
Student ID: 1654583

Project Description

The goals for Assignment 3 are to create an HTTP reverse proxy with the cache. That means that the program will act as an HTTP server but will actually forward received requests to the requested object's origin server in case the object is not cached locally, and then forward the response from the origin server to the original client. In order to avoid sending requests to servers and thus reduce response time and traffic on the network, the proxy cache (or proxy server) will keep copies of objects returned from previous GET requests in a local cache in memory. If a future GET request is made for a cached object, the proxy will return the cached copy — as long as the cached copy is recent enough. Thus the program will act both as a server and a client.

Program logic

For this assignment, the main goal is to modify the handle connection function to take a request from the client(1), send a request to the server(2), receive the response from the server(3), and send a response back to the client(4).

(1) In taking a request from the client, we may treat it as a normal request by parsing the command type, file name, content length, and content, host. We can check if the request is in the cache.

(2)Next, if the request is never seen before, we can make a request to the server to get the requested information. If it is not in the cache, we can send the request(command, file name, HTTP, localhost) to the server. In the case that the file name is already in the cache, we have to send a HEAD request to the server and confirm any changes were made by checking the date modified and maybe content length.

(3) After receiving the information from the server, we parse the response information. If the file name is not seen in the cache before, we put it in the cache as another struct(id, last modified, content, content length). If we have seen the request before, we can check the date modified and make a HEAD request to the server to verify the file size. Before putting items into the cache, we have to check the M flag content size and the c flag maximum files cached size. We can put all this information into a struct for easy access when we start putting stuff into the cache.

(4) Lastly, we send the appropriate response back to the client socket.

Now the main issue is implementing the cache functionality inside the GET function.

After parsing all the previous content, we should check if the filename is in the cache.
If the filename is not in the cache, we send a get request to the server. We parse the time variable.
If the array is not full and the content length is okay, then we place the contents at.
If the array is full and FIFO, then we replace the first index of the array.
If the array is full and LRU, then we replace it the entry with the lowest counter
As a side note, we store the time variable, filename, content size, and content, into the cache as a struct.

If the filename is in the cache, we send a HEAD request to the server.
Use the server response's date and compare it with the cache's time modified.
If the cache is newer or equal then use that file and send the OK back to the client.
If LRU, update the array as the file has recently been used. Else, send a get request back to the server and replace the cache at the original location with the GET request send OK back to the client.

THE PSUEDOCODE AS FOLLOWS:

First, I would check to get the Host variable. I can also get the size of the file using the stat.h struct.

After getting the host variable, I would check the file is inside the cache by looping through the queue array.

```
IF NOT INSIDE THE CACHE:
   Send a request to the server.
   Parse for the time modified and the response length.
   Send the client the header response
   Check for 404 or 400 errors.
   If everything is fine:
      Send the client the content of the response.
      If content size is within m flag size:
          Insert the struct entry to the queue array
          If Insert returns a overflow and FIFO:
              Replace first entry in the array
          If Insert returns a overflow and LRU:
              Replace entry with lowest counter in the array
IF INSIDE THE CACHE:
  Send a head request to the server and parse the time.
  Compare the server time with the queue entry's server time
  If queue entry time is newer or equal:
     Send the client the header response and cache content
  Else:
      Send a get request to the server
      Parse the time from the response from the server
      Send the header response back to the client
      Send the rest of the body back to the client
      Replace the old entry with the new entry at the index
```

Data Structures
For this assignment, the new data structure that I'm using is the cache system. This allows the program to store files for quick access should the need ever come. The cache system comes in either First In First Out (FIFO) or Least Recently Used (LRU). Both of those systems work to provide the proxy with caching abilities.
In addition, I am using the queue data structure to substitute the shared data variable that would have been passed between threads.

Functions
- **clearBuffer(char *buffer)**
  This function takes in a string buffer. It will loop through each index of the buffer and set it to 0, effectively clearing the buffer. Additionally, if used in a nested loop, this function can clear the address space of each index in the buffer.
- **Int insert(struct entry* addItem, queue)**
  This function takes a struct entry and puts it into the queue list. Returns 1 on success, 0 on failure.
- **Int delete(struct entry* addItem, queue)**
  This function takes a struct entry and deletes(dequeue) it into the queue list. Only used for testing purposes. Returns 1 on success, 0 on failure.
- **Int numberOfEntries()**
  This function returns the number of entries in the queue.
- **Void display(queue)**
  This function displays the entire queue. Only used for testing purposes.
- **Struct entry* createList(int size)**
  This function creates a queue of size. Returns the variable of the queue or error on failure.

Questions
Using a large file (e.g. 100 MiB — adjust according to your computer's capacity) and the provided HTTP server:
Start the server with only one thread in the same directory as the large file (so that it can provide it to requests);

> **Start your proxy with no cache and request the file ten times. How long does it take?**

I observe that the proxy with no cache takes a couple of seconds to send a request to the proxy, forward the request to the server from the proxy, parse the response from the proxy, and finally from the proxy, send the response back to the client.

Now stop your proxy and start again, this time with cache enabled for that file. Request the same file ten times. How long does it take?

With cache enabled, there is a noticeable difference between no-cache and cache enabled. It takes about a second or two for the proxy to access its own cache and send a response based on its cache.

Aside from caching, what other uses can you consider for a reverse proxy?

Aside from caching, reverse proxying can allow secure transactions between client and server. Whether it would be an actual eCommerce or just downloading a file safely without delay. The reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

Tests

I mainly tried my program through full system testing. Considering that there are many test cases to consider, I created test cases by myself. These test cases range from error handling to seeing if the files are different from one another. To replicate a client request, I used cURL to send a request to my HTTP proxy program. From there, I can test if my program responds properly to the PUT, GET, and HEAD requests coming from cURL. I can also test if my program receives a response from the server. In the end, if the request spews out a created file or modified file, I can check for those cases using the diff command in the terminal. The program is where it is now through many trials and errors.