Asgn 3 Design Document

Samuel Shin
Cruzid: sayshin

CSE 130, Fall 2019

1 Goal

The goal of this program is to implement caching and logging on top of Assignment 1 and 2. The program should be able to detect the options -c and -l using getopt(). -c is to enable caching and -l [log file] to specify logging. The program should be able to cache files from GET and PUT requests and if a file is already in the cache, then the program should be able to use the files in the cache rather than reading and writing to disk all of the time. If the cache is full, then implement one of the replacement algorithms to decide which file should get deleted and replace with the new file. If the file to be deleted is dirty, then make sure to write to disk before deleting the file from the cache.

2 Assumptions

I'm assuming that the testing will be done with "curl http://localhost:8080 --request-target filename_27_character" for GET requests and "curl -v -T localfile http://localhost:8080 --request-target filename_27_character" for PUT requests. I'm assuming that my GET works fine because on Piazza it says that it's okay if we're getting this warning "Warning: Binary output can mess up your terminal. Use "--output -" to tell Warning: curl to output it to your terminal anyway, or consider "--output Warning: " to save to a file." I'm assuming that the user will input something like 127.0.0.1 for the address name when running ./httpserver [address name] [optional port num] because my program can't handle something like http://localhost. I'm assuming that if using the default port number 80, use sudo ./httpserver [address name] to start the server. I'm assuming that the grader will try the test commands over and over again until it works because sometimes it freezes up (maybe an ubuntu error or something, maybe like 5 times should be okay). I'm also assuming that the files in the cache can be as large as they want to be and that there is no file size limit.

3 Design

The way I am approaching this problem is being able to parse the command line command using getopt(), catch options for -c and -l and be able to save this information in the program. Because I already have logging implemented from the last assignment, I'm going to start with implementing caching right away. Because we are able to implement any caching technique, I'm just doing the simplest way to do caching which is FIFO without second chance. I'm going to create a queue using the c++ deque class and manage a simple queue for my cache that always deletes the first item in the queue if something needs to be replaced. Therefore, I'm only writing back to disk when I need to rather than doing it periodically. As for logging, I can simply add in whether or not the file requested was or wasn't in the cache.

4 Pseudocode

Procedure httpserver

If num of args < 3 || > 6
Error out
Int Option = getopt(cl:) //N has an optional arg and l needs an arg after
If (option == 'c')
Set bool var to true telling that caching has been requested
If (option == 'l')
Set bool var to true telling that log has been requested
If (option == ':' || option == '?')
Something went wrong, handle error

Set port number if port number was given, otherwise default is 80

Create a deque that holds a pair of two strings one for the filename and the other for the file content. //this is the cache

Have a vector of size 4 that contains bools initialized to false that represents if the element in the cache is dirty and needs to write back to disk.

While (true)
Call socket(), setsockopt(), bind(), listen(), accept() in this order to establish connection
recv() request(s) from client

Parse http header

```
If header has GET //if a GET request
    Parse and check if filename is legal
    If caching is enabled
        Check if the filename requested is in the cache
        If it's in the cache
            Simply send appropriate data to the client
            Log in cache
        If it's not in the cache
            Check if the cache is full
            If cache is full
                Check if the first element in the cache is dirty
                If dirty
                    Write to disk and delete
                Read from disk and append file to end of cache
            Else
                Read from disk and append file to end of cache
            Log not in cache
    Else
        Read from disk


Else If header has PUT
    Parse and check if filename is legal
    If caching is enabled
        If file is in the cache
            If the content is different
                Replace the cache data
                Make dirty
            Log in cache
        Else
            If cache is full
                Write dirty file
                Append new file to cache
                Make dirty
                Log not in cache
            Else
                Append new file to cache
                Make dirty
```

Log not in cache

Else

Write to disk

Else

Send a error 500 code