Design Document

Ricky Zhao
CruzID: rjzhao
CSE 130, Fall 2019

1 Goal

The goal of this assignment is to create an httpserver with a cache. The cache will have a capacity of the contents of 4 files. The server will also support logging and will indicate if each files is in the cache or not.

2 Assumptions

In this assignment I will be removing the multithreading component from assignment2 since we are told that it is not a function we will be needing. Like asgn1 & asgn2, I am assumming that the only request we are handling is a GET and PUT request, so any other requests are not valid. I am also assuming that every single PUT request will alter the cache, so each PUT request to the cache will result in a dirty page.

3 Design

Similar to Assignment 2, at the beginning we will check what options was specified by the users of the program. The difference is that we will check for the flag -c instead of -N. We have a global boolean cache_on, indicating whether or not we will be using the cache. We also initialized a global string array cache and cache_filename with 4 elements, representing the cache with the file contents and the filenames in the cache respectively. We will also have a global variable cache_size keeping track how many items are in the cache. We will also have a global int array called dirty_bit to keep track which file contents have been altered. We will also have a bool cache_on, indicating whether or not we will be using the cache.

Cache: For our Cache, we will be implementing the FIFO algorithm for the page replacement algorithm. If the -c flag is specified, we will create a cache of size 4.

For a GET request, I will have a function to check whether or not the file is in the cache. If the file is in the cache, I will send the file content in the cache. If the file is not in the cache, I will check if the cache is full. If the cache is full, I will replace the first item of the cache

with the file content of the requested file. If the first item of the cache is dirty, I will write it back to the disk before removing it from the cache.

For a PUT request, we will first recv data from the client socket. I will then check if the file is in the cache. If it is in the cache, I will store the received data to where that file is in the cache. I will also change the dirty bit of that file to 1. If it is not in the cache, I will check if the cache is full or not. If the cache is not full, I will save the received data to the next free entry in the cache. If the cache is full, I will replace the first item in the cache with the received data. I will also check if the first item is dirty before replacing the item in the cache. If the item is dirty, I will write the content in the cache before removing it. Lastly, I will set the dirty bit of the first item to 1.

Logging: For the design of logging, it will be very similar to asgn2. We will still have the three shared variable, a bool log_write, a char* logfile, and an int offset. However unlike, we will not be calculating the offset before writing to logfile.

At the beginning of the program, if the user specified a -l flag, we will set log_write to true. We will also set the next argument to the char* logfile. For every function, if log_write is true, we will first write the header of the function. If the function fails, we will write a fail header to logfile. If the function succeeds, we will first write the function header.

If the function is a PUT function, we will have a variable called bytecount. We will convert the received data to hex and write the hex to the logfile byte by byte. For every 20 byte of data we receive, we will increase bytecount by 20 and write it to the logfile. Every time we write to the logfile, we will increase the offset by the amount we write.

4 Pseudocode

Httpserver

Global Variable: char* logfile, bool log_write, offset, string cache[4], string cache_filename[4], int dirty_bit[4], bool cache_on, int cache size =0,cache_index

```
Parse_header function ( header,index)
        String word
        While header[i] is not a space
        Word += header[i]
        Return word

 Valid_filename(filename)
        For i=0 to filename.length()
```

```
        If filename[i] is not a-z,A-Z,"-" or "_"
        Return false
        Return true
in_cache(filename)
        Bool in_cache = false
        For i = 0 to cache_size
                If file_name[i]==filename
                        in_cache=true
                        cache_index =i
        Return in_cache


Main function
        Default: port = 80,num_of_thread=4,address
        Getops:
                Case -c:
                        Cache_on = true
                Case -l:
                        logfile=optarg
                        log_write=true
        Extra_arg=0
        While optind <argc
                If extra_arg=0
                        address=argv[optind]
                If extra_arg=1
                        port=argv[optind]
                Extra_arg++
        If argc>7 or extra_arg>2
                exit(fail)

         Create socket
                socket()
                setsockop()
                bind()
                listen()

        while accept()
             recv the header
             parse_header function
             parse_header filename
             Parse_protocol

             if filename contains 27 character and is valid
```

```
    If GET
            get_function(client_socket,filename)
    Else If PUT
            parse_header content length-> content_length
            put_function(client_socket,filename,content_length)
    Else
            send client code 500
            If log_write:
              Pwrite the fail header to logfile
              Offset += length of fail header
    Else
            Send client code 400
            Pwrite the fail header to logfile
              Offset += length of fail header
    Close socket



get_function
     if file does not exist

            If log_write:
                    Pwrite the fail header to logfile
                    Offset += length of fail header

            send code 404 to client
    else if permission denied

            If log_write:
                    Pwrite the fail header to logfile
                    Offset += length of fail header

            send code 403 to client
    else
            If cache_on
                    If in_cache(filename)
                            Send cache[cache_index] to client
                    Else
                            read filename a buffer
                            send buffer to client
                            If cache_size < 4
                                    cache[cache_size]=buffer
                                    Cache_size++
```

```
                    Else
                        If dirty[0] ==1
                            Write cache[0] to disk
                        cache[0]=buffer
                        cache_filename[0]=filename
                        dirty_bit[0] = 0

            If log_write:
                Pwrite the get header to logfile
                Offset += length of get header
                If cache_on
                    If in_cache(filename)
                        pwrite [was in cache] to log file
                        Add to offset
                    Else
                        pwrite [was not in cache] to log file
                        Add to offset
put_fuction
    if file does not exist
        create file
    else if permission denied
        send code 403 to client
        If log_write:
            Pwrite the fail header to logfile
            Offset += length of fail header
    Else
        parse content length
        recv the file of size length from client buffer
        If cache_on
            If in_cache(filename)
                cache[cache_index]=buffer
                dirty_bit [cache_index]=1
            Else
                If cache_size < 4
                    cache[cache_size]=buffer
                    dirty_bit[cache_size]=1
                    Cache_size++
                Else
                    If dirty[0] ==1
                        Write cache[0] to disk
                    cache[0]=buffer
                    cache_filename[0]=filename
                    dirty_bit[0] = 1
```

```
Else
    write buffer to the requested filename

If log_write:
        Pwrite the get header to logfile
        Offset += length of get header
        bytecount = 0 with 8 digits

        Pwrite the PUT header to logfile
        offset+= length of PUT header

        If cache_on
            If in_cache(filename)
                pwrite [was in cache] to log file
                Add to offset
            Else
                pwrite [was not in cache] to log file
                Add to offset

        For every 20 bytes of data from buffer
            Pwrite bytecount
            Convert 20 bytes of buffer in to hex
            Pwrite hex to logfile
            bytecount+=20
```