# Design Document

Ricky Zhao
CruzID: rjzhao
CSE 130, Fall 2019

## 1 Goal

The goal of this assignment is to create an httpserver that supports multiple requests simultaneously. Our httpserver also needs to support logging functionality, which means that our program has to print out a record of each request.

## 2 Assumptions

I am assuming that the functions we are handling is the same as assignment 1, GET and PUT. I am also assuming that we will not be writing to the same file, since the professor stated that we do not need to account for this issue in class.

## 3 Design

At the start running the program, we will use getop to find if -N or -l is specified. If -N is specified, then we will set the following argument as the number of threads we have. If -l is specified, we will set log_write to true and the following argument as the name of the logfile. If there are other arguments that are not following -N or -l, we will set the first one to the address and the second one to the port number. If there are more than 7 arguments or more than 2 extra arguments, then we will throw an error since there are too many arguments. The default number of thread is 4 and port number is 80.

Multi-threading: For the design of we will initialize an array of pthreads to be our pool of threads that we can use. The arguments that are being passed into the threads are the header buffer. I will modify my code from asgn1 so that the GET and PUT functions are thread functions. I will implement a semaphore to keep track of how many threads are active. The semaphore will be initialized to be the number of threads.

Each time a request is passed in, we will call sem_wait, which decrements the semaphore. If the semaphore is equal to 0, that means all the threads are busy, so the semaphore will block. If the semaphore is greater than zero, then we will assign the request to a free thread. After a thread finishes computing a request, we will call sem_post to increment the semaphore. We will also have a shared variable thread_in_use so that we know how many threads are still computing so we do not close the socket before they finish computing. This variable is a critical section and we will use a mutex to lock and unlock when accessing this variable so that it is thread safe.

Logging: For the design of logging we will have 3 shared variable, a log_write boolean, a logfile, and an offset. The boolean log_write indicates if the option -l is specified. If it is we will set it to true, if it is not we will set it to false. The logfile is the name of the file we will log our request to. Both of these will be initialized at the time we execute ./httpserver. Since these two variables will not be changed, they do not require a mutex and is therefore thread safe.

The third variable offset indicates the offset when logging to the logfile. This is a critical region. We will have a mutex called pwrite_mutex. Whenever we modify the offset, we will lock this mutex and when we finish modifying the offset we will unlock the mutex. Since only one thread can access the offset at a time, this is thread safe.

When a thread is processing a request, if the log_write boolean is true, we will log the request to the logfile. We will have a local variable called p_offset. Before logging to the logfile, we will lock the pwrite_mutex. Then we will set p_offset to the offset. Then we will calculate the space we need for logging the request and add it to the offset. After we increased the offset, we will unlock the pwrite_mutex and begin logging using p_offset. Every time we pwrite to loggfile, we add to p_offset instead of directly to offset. Since we already calculated the space we need for logging the request, the request will not be interweaved when logging to the logfile.

## 4 Pseudocode

Httpserver

Global Variable: char* logfile, bool log_write, offset,pwrite_mutex, thread_mutex,
int thread_in_use=0, thread_sem

```
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


Main function
        Default: port = 80,num_of_thread=4,address
        Getops:
                Case -N:
                        Number of thread = optarg
                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()

        Pthread workers[number_of _thread]
        Initizalize thread_sem to number of thread

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

            if filename contains 27 character and is valid
                If GET
                        Sem_wait(thread_sem)
                        Lock thread_mutex
                        thread_in_use++
                        Unlock thread_mutex
                        Find a free thread and pass in void* get_function
                        Pass in filename and new_socket as argument
                Else If PUT
                        Sem_wait(thread_sem)
                        Lock thread_mutex
                        thread_in_use++
                        Unlock thread_mutex
                        Find a free thread and pass in void* put_function
                        Pass in filename, content length and new_socket as argument
                Else
                        send client code 500
                        If log_write:
                                Lock pwrite_mutex
                                P_offset = offset
                                Add to offset
                                Unlock pwrite_mutex
                                Pwrite the fail header to logfile
                Else
                        Send client code 400
                        If log_write:
                                Lock pwrite_mutex
                                P_offset = offset
                                Add to offset
                                Unlock pwrite_mutex
                                Pwrite the fail header to logfile
                Close socket



            void* get_function
                     if file does not exist

                        If log_write:
                                Lock pwrite_mutex
                                P_offset = offset
                                Add to offset
                                Unlock pwrite_mutex
                                Pwrite fail header to logfile

                        send code 404 to client
                else if permission denied
```

```
                If log_write:
                        Lock pwrite_mutex
                        P_offset = offset
                        Add to offset
                        Unlock pwrite_mutex
                        Pwrite fail  header to logfile

                send code 403 to client
        else
        read filename a buffer
          send buffer to client
          If log_write:
                        Lock pwrite_mutex
                        P_offset = offset
                        Add to offset
                        Unlock pwrite_mutex
                        Pwrite the GET header to logfile
        Lock thread_mutex
        thread_in_use --
        Unlock thread mutex
  Void *put_fuction
        if file does not exist
                create file
        else if permission denied
                send code 403 to client
        Else
            parse content length
            recv the file of size length from client buffer
            If log_write:
                        bytecount = 0 with 8 digits
                        Lock pwrite_mutex
                        P_offset = offset
                        Add to offset
                        Unlock pwrite_mutex
                        Pwrite the PUT header to logfile
                        For every 20 bytes of data from buffer
                                Pwrite bytecount
                                Convert 20 bytes of buffer in to hex
                                Pwrite hex to logfile
                                bytecount+=20
            write buffer to the requested filename
        Lock thread_mutex
```

```
thread_in_use --
Unlock thread mutex
```