Program: Multithreaded httpserver Design Document

November 22, 2019

1. **Goal**

   The goal of this program is to build on the previous assignment. Instead of a single-threaded server we are tasked with creating a multithreaded httpserver. The server should be able to handle multiple GET/PUT requests. It should also be able to log each request that comes in through the server. The log will include both the header and the data dumped in hexadecimal format.

2. **Assumptions**

   I'm assuming that the user will execute my program with the curl command specified on the homework attached with comments page. I'm also assuming that the program is going to be tested on the Ubuntu version 18.04.3.

   I'm also assuming that my program will continue until the server is stopped. This is to ensure that the server accepts multiple curl requests. I'm assuming that I will need to use a queue to keep track of the requests. I will also use mutexes to make sure that the socket I pass in won't be used by another thread. I assume that I will need to use wait/signal to ensure the program doesn't use up too much cpu resources.

3. **Design** The general approach I had for this program was to fix my assignment1 first and then work on the multithreading and logging. If I fixed everything in assingment1 then I could focus on multithreading and logging without worrying about errors within that portion of my code. Once, I fixed all my issues I began working on implementing getopt. Using getopt was helpful because now I can take in the flags -lN and not have to worry about the placement of the other two parameters (address and port). Then once this was finished I started working on multithreading. I tried to understand how to implement multithreading from TAs, google, discussion, and piazza.

   From my understanding of multihreading and the addition of my program I will try to explain the questions asked in the assignment document. I believe that my server is thread safe because my server creates the worker threads for my dispatcher thread based on the amount specified in the command line. Then I will have an inifinite loop that monitors incoming connections. If they are accepted then it will put that socket connection in a linked list. Then my dispatcher thread will have an infinite loop that

checks the linked list if it doesn't return null when we dequeue from our list then we will execute doGetPut on the object returned from the linked list which is our socket connection. The critical region here is when we queue and dequeue from our linked list. we don't want threads to have access to the same socket connection. So to make sure we don't access the critical region when we should I used a mutex. For when I queued and dequeued to the linked list I put a mutex lock before it and a mutex unlock after it. Also, I used cond_wait and cond_signal to put the threads to sleep so it doesn't use up all of the CPU resources.

For testing I followed the commands used in the last assignment from grading and the commands given in the assignment 2 with comments. I ran the script to ensure that multithreading was working properly.

4. **Pseudocode** For this pseduocode I will not go into the specifics of the doGetPut function because it was the last assignment and we had to make sure it works before starting on this assignment. So, let's assume that doGetPut handles get/put requests without any issues.

initialize mutex
initialize cond

doGetPut()

dispatcherThread()
while true **do**
    lock mutex
    cond wait
    dequeue from linked list
    call doGetPut() if it doesn't return NULL
end while

Main()
if $server\_fd = socket(params) == 0$ **then**
    print("socket failed")
end if
$address.sin_family = AF_INET$
$address.sin_addr.s_addr = INADDR_ANY$
while getopt()!= -1 **do**
    have a switch case for the flags
end while
Count number of optind to see if only address is provided or address and port is provided
If optind is 1 then default port is 80

```
If optind is 2 then port is whatever was specified
create number of threads specified and call to dispatcher thread
if bind(server, ...) < 0 then
    perror("bind failed")
end if
if listen(server_fd, 3) < 0 then
    perror("listen")
end if
while true do
    if new_socket = accept(server_fd, ...) < 0 then
        perror("accept")
    end if
    create pointer to socket
    lock mutex
    enqueue pointer to linked list
    cond signal
    unlock mutex
end while
```