Sehej Sohal sssohal 1650056

Assignment 1 - HTTP-Server Design Document

1 Goal

The goal of this program is to create and host, from the ground up, a working HTTP server that can accept and serve content via GET and PUT HTTP requests. It should be able to accept and serve any size of file, given enough time, and should not modify the data in any way.

2 Assumptions

- The open/read/write/send/recv system calls have some sort of max amount of bits they can read, regardless of buffer size, and that size differs between OS's. Meaning that to have the code work on multiple OS's, I need to keep the buffer size small.
- Usage of common HTTP paradigms, such as pausing and expecting a 100 Continue before uploading a large file.
- Maximum buffer sizes are 32kb, but we're welcome to use smaller buffers.
- In order to avoid overcomplicating code, functionality has been split up into one object, called HTTP server. This object will handle everything an HTTP server should, and nothing more

3 Design

The general approach I'm taking to this assignment is as such: first, set up and bind a socket to an IP address and a port. In terms of data structures, I'm only really using a C++ style class, char arrays, and strings. Complex data structures are not needed for this assignment, but may be needed in further extensions.

The IP address and Port can both be user specified, as per the assignment specification. As I said before, I discovered in assignment0 that not all read/recv's are the same, so I'm planning on keeping my buffer small, at 4kiB to avoid any unnecessary errors

Once the socket is created and has an address bound to it, we will start an infinite loop with a while(INT) that listens and accepts incoming connections. To stop this loop, the user simply will need to input a CTRL-C on their keyboard.

Once a connection has been accepted, it will be passed off to a serve function. My reasoning behind this is that: if we will need to expand this assignment in the future to be able to handle multiple requests at once, it will be easier to do so if we can simply queue up the file descriptor returned by the accept function.

The serve function is where a bulk of the code is. The serve function will parse the HTTP header to retrieve the method, filename(if applicable), content-length(if applicable), any expectations(if applicable) and the content-body(if applicable) with the C++ regex library. I'm purposefully avoid abstracting the received HTTP request into it's own object as working with char pointers and functions is a little tricky when you want to run memory safe, so I'm trying to avoid this if at all possible. In the future, this may change if it becomes too complicated. After the request is parsed, I will have some sort of case-by-case function on the received method, to handle GETs, PUTs, and any other HTTP methods. This should be easily extendable, for upgradability.

Once the HTTP request is handled, the client file descriptor will be closed, and the program will loop back around, to an accept state, where it waits for another incoming connection.

The main function will contain argument handling for setting up the server with user described ports and addresses, and the http_server class will have a plethora of private utility functions.

document continued on next page

4 Pseudocode

```
http_server.h/cpp

class httpserver{
    public:

        httpserver(port(optional), address(optional)) {
            get socket file descriptor
            bind socket(port, address)
        }

        listen(port) {
            while(infinite) {
                listen()
                if int client = accept()
                    serve(client)
            }
            close(port)
        }

    private:
        address = default 127.0.0.1
        port = default 8080

        echo(function from assign0 to echo from 1 fd to another fd)

        serve(file descriptor) {
            buffer = 4kib
            recv(client fd, buffer, 4kib)
            parse recv with util functions(getMethod, getFilename,
                                           getContentLength, getExpectation,
                                           getBody)
            if(httpMethod == get) {
                get(request info)
            } else if (httpMethod == put) {
                put(request info)
            } else {
                send 403 bad request
            }
        }

        get(http request) {
            check filename
            fd = open(filename)
            *check if fd < 0 for 500 internal error*
                or 400 bad request
                or 403 forbidden
                or 404 not found
            echo(fd, client)
```

```
            close(client)
            200 OK
        }

        put(http request) {
            check filename
            fd = open(filename)
            *check if fd < 0 for 500 internal error*
                or 400 bad request
                or 403 forbidden
            echo(client, fd)
            close(client)
            201 CREATED
        }

        util functions:

        getMethod - gets method
        getFilename - gets and validates filename
        getContentLength - gets and validates length
        getContentType - gets and validates type
        getExpectation - gets expectations if any
        getBody - gets content body

}

main.cpp

int main(argc, argv) {

    check if there's more than 1 argument
    check if arg is port or address

    if port and not address {
        httpserver(port)
    } else if address and not port {
        throw error(c++ does not allow the first parameter to be optional)
    } else if address and port {
        httpserver(port, address)
    } else {
        httpserver(default, default)
    }

    httpserver.listen()
}
```