Design Document: Simple Web Server v1.1

Robert Hu

CruzID: ryhu

1.0 Introduction

1.1 Goals and objectives

The overall goal of this program is to create a single threaded simple HTTP web server with very limited scope.

1.2 Statement of scope

The web server will only handle GET, HEAD, and PUT requests and will only care about the HTTP version, the command, the file, and the size of the contents if needed. Only some response codes will be implemented namely: 200 OK, 201 Created, 400 Bad Request, 403 Forbidden, 404 Not Found, 500 Internal Server Error. File names must follow certain specifications; they must be less than or equal to 27 characters with these characters limited to being either a dash, an underscore, a lower/uppercase letter, and a number.

2.0 Data Design

2.1 Internal software data structure

A buffer is implemented to handle message passing within the main program to handle incoming messages.

2.2 Global data structures

A struct is implemented to handle the processing of messages coming from the client socket and the split the important information in the message to various vars.

2.3 Database description

The buffer is using a predefined size of 4 Kib for its overall length. For GET requests, the program reads from the requested file into the buffer and writes to the client socket. For PUT requests, the program reads from the client socket into the buffer and writes to a file.

The struct, httpObject, contains 6 vars. The first three vars are char[] to contain the method, the filename, and the httpversion. The following two vars are a ssize_t and an int to contain the content length and future status code. The last var is a uint8_t buffer with size of 4 Kib.

3.0 Architectural and component-level design

3.1 System Structure

The design is split into various different functions. The main functions are read_http_response, process_request, and construct_http_response. The first function reads in the http response from the client socket and scans and splits the response into

the info we needed for the rest of the program. The second function processes the message that was received. The third function constructs a response to send back to the client socket. All three of these functions take in the client socket and the message struct as inputs.

3.2 Description of Main helper function 1

3.2.1 Main helper function 1 Responsibility

This function is responsible for receiving a message from the connected client socket and grabbing from the message the needed information. The function takes in the socket descriptor for the client socket as well as the httpObject struct to contain the message info. The function also checks and ensures the file names fit the specification.

3.2.2 Main helper function 1 Algorithm

INPUT: client socket, httpObject message
OUTPUT: none

1. Receive client request into buffer
2. sscanf first line to get method, filename, and httpversion
3. Check if first char in filename is '/'
    a. if it is, remove the '/'
    b. otherwise, set status code to 400
4. Check if all filename characters are valid and total is 27 or less
    a. if the filename doesn't satisfy these conditions, set status code to 400
5. strtok to split request header with delimiters " \r\n"
6. Find "Content-Length:" and retrieve the content length if it exists

3.3 Description of Main helper function 2

3.3.1 Main helper function 1 Responsibility

This function is responsible for processing the message to grab additional information as well as checking some errors. The function opens files, grabbing the content length for head and get commands as well as checking the possible errors that can occur such as 403 and 404 codes.

3.3.2 Main helper function 1 Algorithm

INPUT: client socket, httpObject message
OUTPUT: none

1. Check if status code has been set to 400
    a. If it has, skip this function
2. FOR the each method
    a. Open the file with the appropriate flags

        i. If opening fails, compare error and set status code appropriately and return
    b. Set status code on success
        i. 201 for PUT and 200 for GET and HEAD
    c. For PUT, write request body into filename
    d. For GET and HEAD, grab content length with fstat

3.4 Description of Main helper function 3

3.4.1 Main helper function 1 Responsibility

This function is responsible for creating the http response to send back to the client socket. This function will get the file data for GET requests and send it to the client socket. This function also gets the contents of PUT requests and creates/writes to the named file. This is also the function that creates the response header based on the httpObject, message.

3.4.2 Main helper function 1 Algorithm

INPUT: client socket, httpObject message
OUTPUT: none

1. Switch statement to compare and create status strings for each status code
    a. 200 -> OK
    b. 201 -> Created
    c. 400 -> Bad Request
    d. 403 -> Forbidden
    e. 404 -> Not Found
    f. 500 -> Internal Server Error
2. Create response header using httpObject
3. If status code is not 200 or 201, return
4. Otherwise
    a. For GET, write filename contents to client socket

4.0 Change log

The main change in this version is moving the PUT file writing from the 3rd function over to the 2nd function. Some minor changes include additional error handling.