Asgn1 Design Document

Jessica Pan

CruzID: jeypan

CSE130, Fall 2019

1. Goal

The goal of this assignment is to implement a program that functions as a simple-threaded HTTP server. The server will only respond to a simple GET and PUT commands to read and write "files" names with exactly 27-character ASCII names. The server is to persistently store files in a directory on the server (which is the current directory the server's on). This way the server can be restarted or run on a directory that already has files. If the error encounters any errors, it will respond to the client with a proper response.

2. Assumptions

The assumptions I'm making for this program is when a GET request is made, the server will understand the request and print the data corresponding to the file name specified in the request (assuming that the file name satisfies the file name criteria). I'm also assuming, since clients, a web browser, making GET requests from HTTP servers don't store the pages received into their own directory, therefore, my HTTP server will not store the file requested onto the client's directory. For a GET request, the server will simply write the data to the client's standard output.

Next is when client sends the PUT request, if the file isn't able to be accessible by curl or if the file isn't present in the client's directory for curl to call it to PUT into the server, then curl would, on its own, disconnect the client-server connection with an error of `curl: can't open 'filename'` when it's ran with the command: `curl --upload-file filename [ip address]:[port #]` or `curl -T filename [IP address]:[port #]`.

Another assumption I'll be making is that when the server is binded to an IP address, the client must call the same IP address and the same port to the server in order to make the proper connection to the server. For example, if the server were to set it's IP address to 127.0.0.2 with port 8080, then the client must call the same IP address and port number: `curl 127.0.0.2:8080`.

3. Design

The approach I'm taking to this program is to first check the arguments when the user starts to run the program. From the arguments, there must be an IP address

specified and an optional port number. If the port number isn't specified, then the standard port number will be set, which is 80. After setting the port number to a variable, the socket file descriptor is created. If program is unable to create socket file descriptor, program will exit with exit failure. Once the socket file descriptor is created, the next task was to define the IP address and port number specified in the arguments when the program was ran. As we define the IP address, we need to make sure to check if the user used an alias for the IP address (i.e. localhost). If the user did use an alias for the IP address, in this case "localhost," then the program will convert it to "`127.0.0.1`" and set that as the IP address.

After the IP address and port number is set, the program will bind the address to the server using the function, `bind()`. The server will then listen for the socket using the function, `listen()`. If these two functions return -1, then the program will exit with EXIT_FAILURE. If not, then the program will create a connection. To maintain this connection, for the server to be constantly listening, there will be a while loop that'll always be true, `while(1)`.

Once the program is in the while loop, the program will wait for a connection and accept a connection with the client with the function `accept()`. This function will only accept clients specifying the same IP address and port number as the server in its GET/PUT requests. If this function returns a -1, then the program will exit with EXIT_FAILURE. Otherwise, the program will continue to parse the GET/PUT header received from a client.

The program will parse the header of the request with the function `strtok()` with the delimiter being "`\r\n`." By doing this, it'll place each line in the header into a char array of characters. Then, the program will parse each line with the same function but with a space delimiter. The program will then `push_back()` the values of the header to its corresponding key in a vector of strings.

After storing all the values of the header, the program will then begin to check whether the file specified by the client is valid. For the file name to be valid, it needs to be exactly 27-ASCII characters, but these characters must consist of the characters '`A`' to '`Z`', '`a`' to '`z`', '–', and '`_`'. Any other characters that don't fall within this range will cause the program to send the client a status code response of 400 (Bad Request) and then the server will wait for another connection from another client.

If the file name fits the specified criteria, then the program will check what method is used for the client's request. If their request is not a GET or PUT, then the program would respond to the client with a status code of 500 (Internal Server Error). The server would then wait for the next connection made from a client.

If the method is a GET and the file name is valid, then the program will open this server with the function `open()`. Depending on the value returned by `open()`, if it's -1, then the program will send a 403 status code (for accessing a forbidden file on the server) or a 404 status code (if the file doesn't exist on the server). If the returned value is not -1, then the server will `send()` a 200 OK status code and the content length of the file, also it'll `write()` the data of the file to the client's standard output after reading the data with `read()`. This GET method does have one exception. That is if there's no file name specified by the client, then the server will write no data to the client's standard output and `send()` a 200 OK status code because the client isn't requesting any data. Once this is done, the program will wait for the next connection and request from a client.

If the method is a PUT and the file name is valid, then the program will then check if the file being placed in the server is forbidden (i.e. if the client has permission of the file to place into the server). If it's forbidden then a status code of 403 will be sent to the client. If the file that the client is trying to put to the server doesn't exist in the client's directory then my program would return a 400-status code. Except if the client were to put a file to the server that didn't exist in the client's directory `curl()` would stop the client from making this request.

Once the file being put is valid, the program will check if the content length in the PUT header was specified. The results depend on the specification of the content length. If the content length isn't specified, then the server will send the client a status code of 201 (Created) and continue to read the data from client and write the data to a newly created file on the server's directory. The server will continue to `read()` until the client closes the connection by typing ctrl+c in the client's terminal. If the content length is specified then the program will send a 201 (Created) status code and write the data of content length into the newly created file into the server's directory.

The very end, before the while loop ends, the program will close the socket connection between the server and client and then create a new socket once the while loop loops back around for the next connection.

4. Pseudocode

Below is the pseudocode for the httpserver program.

procedure httpserver

    if argc == 3 **then**
        port ← $argv_2$
    else if argc ==2 **then**
        port ← 80
    else
        WARNX("Usage: ./httpserver [IP address] [optional port #]

```
        exit(1)
end if

addrlen ← sizeof(address)
if (server_fd ← SOCKET(domain, type, protocol)) == 0 then
        PERROR("err in socket function")
        EXIT(EXIT_FAILURE)
end if
#define PORT port
address.sin_family ← domain
if argv1 == "localhost" then
        address.sin_addr.s_addr ← INET_ADDR("127.0.0.1")
else
        address.sin_addr.s_addr ← INET_ADDR(argv1)
end else
if BIND(server_fd, address, sizeof(address)) < 0 then
        PERROR("err in bind function")
        EXIT(EXIT_FAILURE)
end if
if LISTEN(server_fd, backlog) < 0 then
        PERROR("err in listen function")
        EXIT(EXIT_FAILURE)
end if
while (1) do
        if (new_socket ← ACCEPT(server_fd, address, addrlen)) < 0 then
                PERROR("err in accept function")
                EXIT(EXIT_FAILURE)
        end if
        Declare buffer to size 4000
        Declare read_buf[] to size 4000
        i ←0
        valread ← READ(new_socket, buffer, sizeof(buffer))

        token ← STRTOK(buffer, "\r\n")
        while token != null do
                read_buf[i++] ← token
                token ← STRTOK(null, "\r\n")
        end while
```

```
enum HeaderKey { Method, Filename, Protocol, Host, Useragent, Accept,
ContentLen, Expect }
Create struct S of a char array values
Create vector of type S called Headers

tok ← STRTOK(read_buf[Method], " ")
while tok != null do
    if tok == "GET" or tok == "PUT" then
        s.values ← tok
        Headers.PUSH_BACK(s)
    else if tok != "HTTP/1.1" then
        s.values ← BASENAME(tok)
        Headers.PUSH_BACK(s)
    else
        s.values ← tok
        Headers.PUSH_BACK(s)
    end if
    tok ← STRTOK(null, " ")
end while
if Headers[Method].values == "GET" then
    t ← STRTOK(read_buf1, " ")
    while t != null do
        if t != "Host:" then
            s.values ← t
            Headers.PUSH_BACK(s)
        end if
        t ← STRTOK(null, " ")
    end while
    t ← STRTOK(read_buf2, " ")
    while t != null do
        if t != "User-Agent:" then
            s.values ← t
            Headers.PUSH_BACK(s)
        end if
        t ← STRTOK(null, " ")
    end while
    t ← STRTOK(read_buf3, " ")
```

```
        while t != null do
            if t != "Accept:" then
                s.values ← t
                Headers.PUSH_BACK(s)
            end if
            t ← STRTOK(null, " ")
        end while
    else if Headers[Method].vlaues == "PUT" then
        for i ← 1; read_buf[i] != null; ++i do
            t ← STRTOK(read_buf_i, " ")
            while t != null do
                if i == 1 and t != "Host:" then
                    s.values ← t
                    Headers.PUSH_BACK(s)
                end if
                if i == 2 and t != "User-Agent:" then
                    s.values ← t
                    Headers.PUSH_BACK(s)
                end if
                if i == 3 and t != "Accept:" then
                    s.values ← t
                    Headers.PUSH_BACK(s)
                end if
                if i == 4 and t == "Expect:" then
                    eh[] ← "None"
                    s.values ← t
                    Headers.PUSH_BACK(s)
                else if i == 4 and t != "Content-Length:" then
                    s.values ← t
                    Headers.PUSH_BACK(s)
                else if i == 5 and t != "Expect:" then
                    s.values ← t
                    Headers.PUSH_BACK(s)
                end if
                t ← STRTOK(null, " ")
            end while
        end for loop
    else
```

```
                    isrErr[] ← “HTTP/1.1 500 Internal Server Error\r\nContent-Length:
0\r\n\r\n"
                    SEND(new_socket, isrErr, strlen(isrErr), 0)
                end if

                validFname ← false
                fname ← Headers[Filename].values
                if fname != 27 then
                    validFname ← false
                else
                    for k ← 0; fname_k != ‘\0’; ++k do
                        if ‘A’ <= fname[k] <= ‘Z’ or ‘a’ <= fname[k] <= ‘z’ or ‘0’ <=
fname[k] <= ‘9’ or fname[k] == ‘-’ or fname[k] == ‘_’ then
                            validFname ← true
                        else
                            validFname ← false
                        end if
                    end for loop
                end if
                if validFname == false then
                    notTs[] ← "HTTP/1.1 400 Bad Request\r\nContent-Length:
0\r\n\r\n"
                    SEND(new_socket, notTs, strlen(notTs), 0)
                end if
                if validFname == true and Headers[Method].values == “GET” then
                    Declare buffy to the size of 4000
                    nbytes ← sizeof(buffy)
                    filepath ← Headers[Filename].values
                    if (pfd ← OPEN(filepath, O_RDONLY, 0)) == -1 then
                        Create struct stat s
                        if STAT(filepath, &s) == 0 then
                            k ← s.st_mode
                            if (k & S_IRUSR) == 0 then
                                getErr[] ← "HTTP/1.1 403
Forbidden\r\nContent-Length: 0\r\n\r\n"
                                SEND(new_socket, getErr, strlen(getErr), 0)
                            end if
                        else
```

```
                                        getErr[] ← "HTTP/1.1 404 Not Found\r\nContent-
Length: 0\r\n\r\n"
                                        SEND(new_socket, getErr, strlen(getErr), 0)
                                end if
                        else
                                while (bytes_read ← READ(pfd, buffy, nbytes)) >= 1 do
                                        Declare l with size of 4
                                        SPRINTF(l, "%zu", bytes_read)
                                        resp[] ← "HTTP/1.1 200 OK\r\nContent-Length: "
                                        STRCAT(resp, l)
                                        STRCAT(resp, "\r\n\r\n")
                                        SEND(new_socket, resp, strlen(resp), 0)
                                        WRITE(new_socket, buffy, bytes_read)
                                end while
                        end if
                        CLOSE(pfd)
                end if

                if validFname == true and Headers[Method].values == "PUT" then
                        Declare patty to size of 29
                        STRCAT(patty, Headers[Filename].values)
                        filename ← patty
                        if (pfd ← OPEN(patty, O_WRONLY | O_CREAT | O_TRUNC, 0666))
== -1 then
                                Create struct stat s
                                if STAT(patty, &s) == 0 then
                                        k ← s.st_mode
                                        if (k & S_IRUSR) == 0 then
                                                getErr[] ← "HTTP/1.1 403
Forbidden\r\nContent-Length: 0\r\n\r\n"
                                                SEND(new_socket, getErr, strlen(getErr), 0)
                                        end if
                                else
                                        putInval[] ← "HTTP/1.1 400 Bad
Request\r\nContent-Length: 0\r\n\r\n"
                                        SEND(new_socket, putInval, strlen(putInval), 0)
                                end if
                        else
                                Declare datta to size of 32000
```

```
                                    if “None” == Headers[ContentLen].values then
                                        resp[] ← "HTTP/1.1 201 Created\r\nContent-
Length: 6\r\n\r\n"
                                        SEND(new_socket, resp, strlen(resp), 0)
                                        while (rdata ← READ(new_socket, datta,
sizeof(datta))) >= 1 do
                                            WRITE(pfd, datta, rdata)
                                        end while
                                    else
                                        if (rdata ← READ(new_socket, datta,
sizeof(datta))) >= 1 then
                                            WRITE(pfd, datta, rdata);
                                        end if
                                        Declare l to size of 4
                                        SPRINTF(l, “%s”, Headers[ContentLen].values)
                                        resp[] ← "HTTP/1.1 201 Created\r\nContent-
Length: ”
                                        STRCAT(resp, l)
                                        STRCAT(resp, “\r\n\r\n”)
                                        SEND(new_socket, resp, strlen(resp), 0)
                                    end if
                                end if
                                MEMSET(patty, 0, sizeof(patty))
                                filename = null
                                CLOSE(pfd)
                            end if
                            CLOSE(new_socket)
                        end while
                        return 0
                    end procedure
```