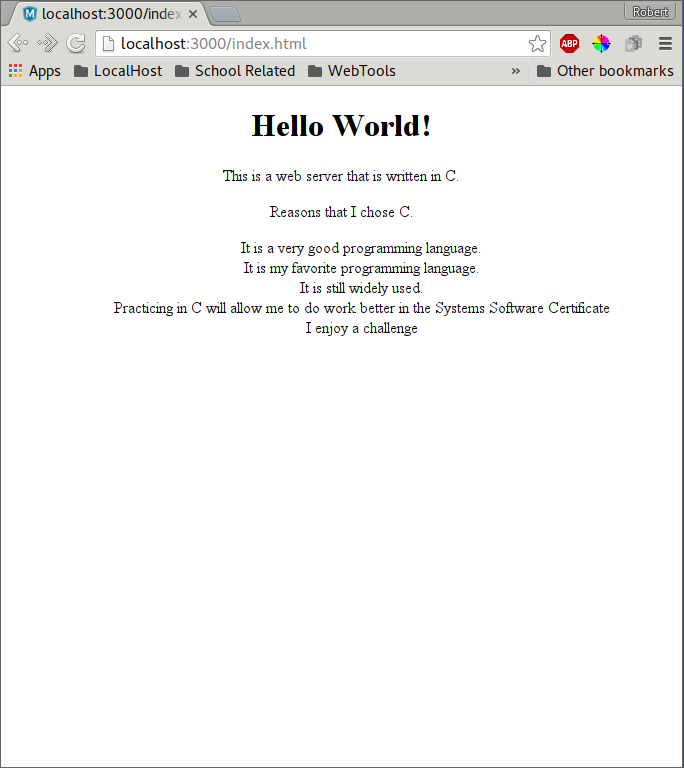
# Socket Programming Assignment 2: Web Server

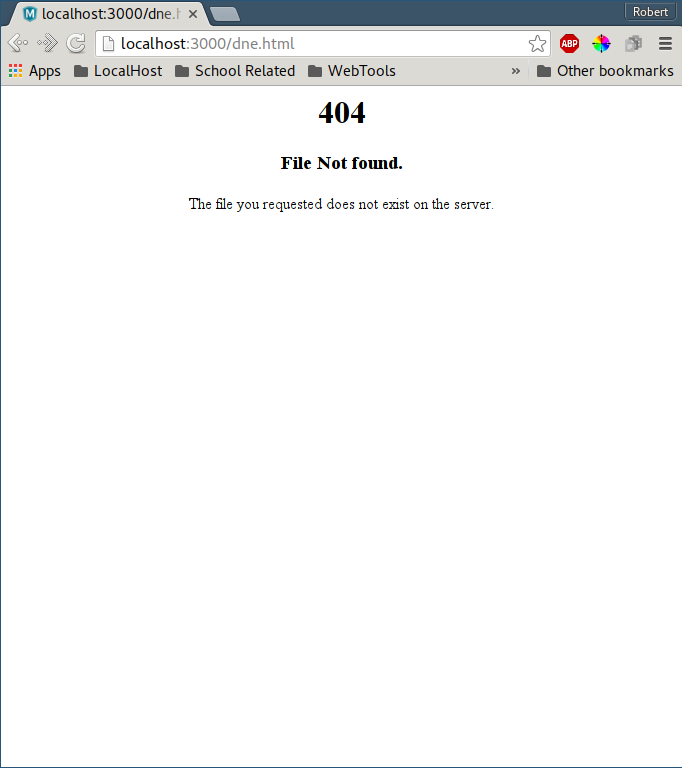
The html file output by the webserver and parsed and rendered by Google Chrome.



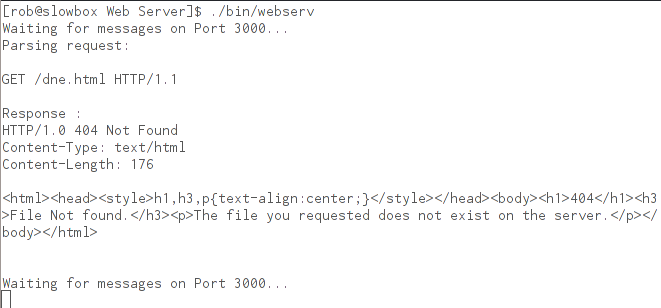
The terminal running the webserver with active response output for debugging/visual acknowledgement of the response message. Outputs the request and the response.



The 404 Message rendered by chrome that was received from the web server.



The terminal activity that was running the web server process. Outputs the request and the response.



## include/webserv.h

The header file source code for the web server.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <time.h>

#include <math.h>

#define BUFF\_SIZE 2048

#define ERR(msg){perror(msg);exit(1);}

//Define a server port to access from client, and init on server

#define SERVER\_PORT 3000

#define SERVER "localhost"

/\* Request Struct \*/

struct http\_request{

char \*request\_method;

char \*http\_version;

char \*request;

char \*content\_type;

char \*content\_len;

char \*serv;

char \*date;

};

/\* Response Struct \*/

struct http\_response{

int \*status\_code;

char \*status;

char \*date;

char \*file\_contents;

char \*content\_length;

int \*content\_length\_int;

char \*full\_response;

};

/\* Static 404 message\*/

static char\* not\_found\_response\_template =

"HTTP/1.0 404 Not Found\r\n"

"Content-Type: text/html\r\n"

"Content-Length: 176\r\n"

"\r\n"

"<html><head><style>h1,h3,p{text-align:center;}</style></head><body><h1>404</h1><h3>File Not found.</h3><p>The file you requested does not exist on the server.</p></body></html>"

"\r\n"

"\r\n";

/\* Static 200 start of response message\*/

static char \*two\_hundred\_ok =

"HTTP/1.1 200 OK"

"\r\n"

"Server: webserv/1.0"

"\r\n";

void parseRequest(char \*buffer);

void buildResponseDate();

void buildFullResponse();

int getFileContents();

int responseLen();

## src/webserv.c

The source code for the web server.

#ifndef WS\_H

#define WS\_H

#include"webserv.h"

#endif

struct http\_request request\_header;

struct http\_response response\_header;

int main(int argc, char \*argv[])

{

unsigned int sockfd, newsockfd, clilen;

char buffer[BUFF\_SIZE];

struct sockaddr\_in server\_addr, cli\_addr;

int n;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

ERR("Error Opening Socket");

/\* Reset Memory \*/

memset((char \*)&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

int port = SERVER\_PORT;

server\_addr.sin\_port = htons(port);

if (bind(sockfd, (struct sockaddr \*)

&server\_addr,sizeof(server\_addr)) < 0)

ERR("Error Binding Socket");

while(1){

printf("Waiting for messages on Port %d...\n",SERVER\_PORT);

listen(sockfd,5);

clilen = sizeof(cli\_addr);

newsockfd =

accept(sockfd, (struct sockaddr \*) &cli\_addr, &clilen);

if (newsockfd < 0)

ERR("Error Accepting Message");

memset((char \*)&buffer, 0, BUFF\_SIZE);

n = read(newsockfd,buffer,255);

parseRequest((char\*)&buffer);

char \*response = NULL;

/\* Reset Memory \*/

memset(&response, 0, sizeof(request\_header.request)); if(getFileContents(request\_header.request)){

buildResponseDate();

buildFullResponse();

response = response\_header.full\_response;

}else{

response = not\_found\_response\_template;

}

response\_header.full\_response = NULL;

printf("Response : %s\n",response);

n = write(newsockfd,response,strlen(response));

close(newsockfd);

if (n < 0)

ERR("Error Responding");

}

return 0;

}

/\*\*

\* Parse the incoming TCP request.

\* Special case first line, strtok by space.

\* Strtok by : after first line.

\*/

void parseRequest(char \*buffer){

char \*end\_str;

char \*token = strtok\_r(buffer,"\r\n",&end\_str);

int method\_set = 0;

int first\_line = 1;

while(token != NULL){

/\* While there are tokens in "string" \*/

char new[strlen(token)];

char \*end\_token;

char \*end\_token2;

int strln = strlen(token);

strncpy( new, token, strlen(token) );

new[strln]='\0';

char \*tok2,\*tok3;

if(first\_line){

tok2 = strtok\_r(new," ",&end\_token);

first\_line = 0;

tok3 = NULL;

}else{

tok3 = strtok\_r(new,":",&end\_token2);

tok2 = NULL;

}

while(tok2 != NULL){

if(!method\_set && (strcmp(tok2,"GET") == 0

|| strcmp(tok2,"POST") == 0

|| strcmp(tok2,"PUT") == 0

|| strcmp(tok2,"DELETE") == 0)){

request\_header.request\_method =

malloc(strlen(tok2) \* sizeof(char));

strcpy(request\_header.request\_method, tok2);

method\_set = 1;

}

tok2 = strtok\_r( NULL, " ",&end\_token);

request\_header.request =

malloc(strlen(tok2) \* sizeof(char));

strcpy(request\_header.request, ++tok2);

tok2 = strtok\_r( NULL, " ",&end\_token);

request\_header.http\_version =

malloc(strlen(tok2) \* sizeof(char));

strcpy(request\_header.http\_version, tok2);

tok2 = NULL;

}

while(tok3 != NULL){

tok3 = strtok\_r( NULL, ":",&end\_token2);

}

/\* Get next token: \*/

token = strtok\_r( NULL, "\r\n",&end\_str);

}

}

/\*\*

\* Build proper TCP Formatted date.

\*/

void buildResponseDate(){

char time\_buf[1000];

time\_t now = time(0);

struct tm tm = \*gmtime(&now);

strftime(time\_buf,sizeof time\_buf, "Date: %a %d %b %Y %H:%M:%S %Z\r\n", &tm);

response\_header.date = malloc(sizeof(time\_buf));

response\_header.date = time\_buf;

}

/\*\*

\* Get the requested file contents, and also get

\* the content length of the data file.

\*/

int getFileContents(char \*filename){

unsigned long length;

FILE \*fp = fopen(filename,"rb");

if(fp){

fseek(fp,0,SEEK\_END);

length = ftell(fp);

fseek(fp,0,SEEK\_SET);

response\_header.file\_contents = malloc(length);

if(response\_header.file\_contents){

fread(response\_header.file\_contents,1,length,fp);

char cl[2048];

strcpy(cl,"Content-Length: ");

char int\_len[(int)floor(log10(abs(length)))+1];

sprintf(int\_len,"%lu",length);

strcat(cl,int\_len);

strcat(cl,"\r\n");

strcat(cl,"\0");

response\_header.content\_length = malloc(sizeof(cl));

response\_header.content\_length = cl;

response\_header.content\_length\_int = (int\*)length;

return 1;

}

}

return 0;

}

/\*\*

\* Build Full Response including headers to send to requester

\*/

void buildFullResponse(){

char final\_form[responseLen()+100];

final\_form[0] = '\0';

final\_form[responseLen()+100] = '\0';

strcat(final\_form,two\_hundred\_ok);

strcat(final\_form,response\_header.date);

strcat(final\_form,response\_header.content\_length);

strcat(final\_form,"\r\n");

strcat(final\_form,response\_header.file\_contents);

strcat(final\_form,"\r\n");

strcat(final\_form,"\r\n");

response\_header.full\_response = malloc(sizeof(final\_form));

/\* Reset Memory \*/

memset(response\_header.full\_response, 0, sizeof(final\_form)); strcpy(response\_header.full\_response,final\_form);

}

/\*\*

\* Get Full Response length to attach to header

\*/

int responseLen(){

int ok = strlen(two\_hundred\_ok);

int date = strlen(response\_header.date);

int cl = strlen(response\_header.content\_length);

int cll = (int)response\_header.content\_length\_int;

return ((ok+date+cl)\*sizeof(char))+cll;

}