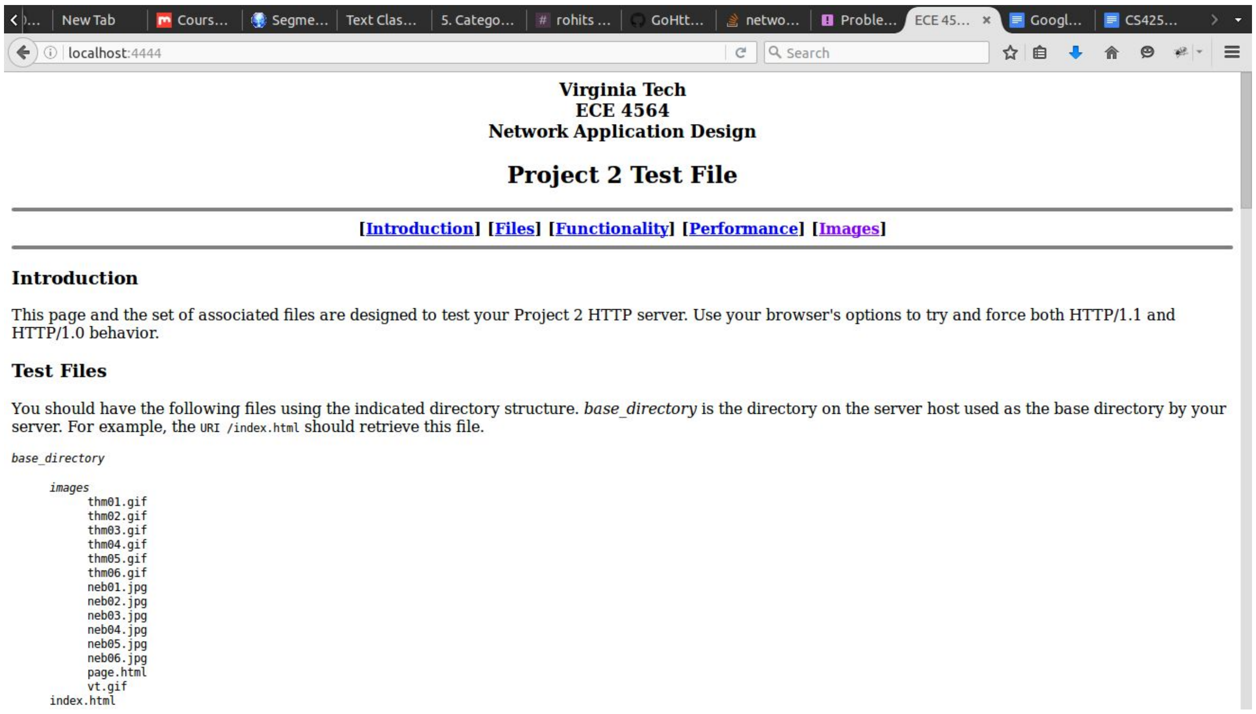
CS425: Computer Networks
IIT Kanpur
Project1: Designing a HTTP-Server
Date: Fri, Aug 19

Name: Ashwani Kumar Gautam
Roll: 13165
Email: ashwing@iitk.ac.in

List of implemented options:

1. Allowed the server port to be initialized at start-up, via command line
2. Allowed the server base_directory to be specified from conf. file
3. Include the Date and Server fields in the Response message headers

Screenshots:



Virginia Tech
ECE 4564
Network Application Design

Project 2 Test File

[Introduction] [Files] [Functionality] [Performance] [Images]

Introduction

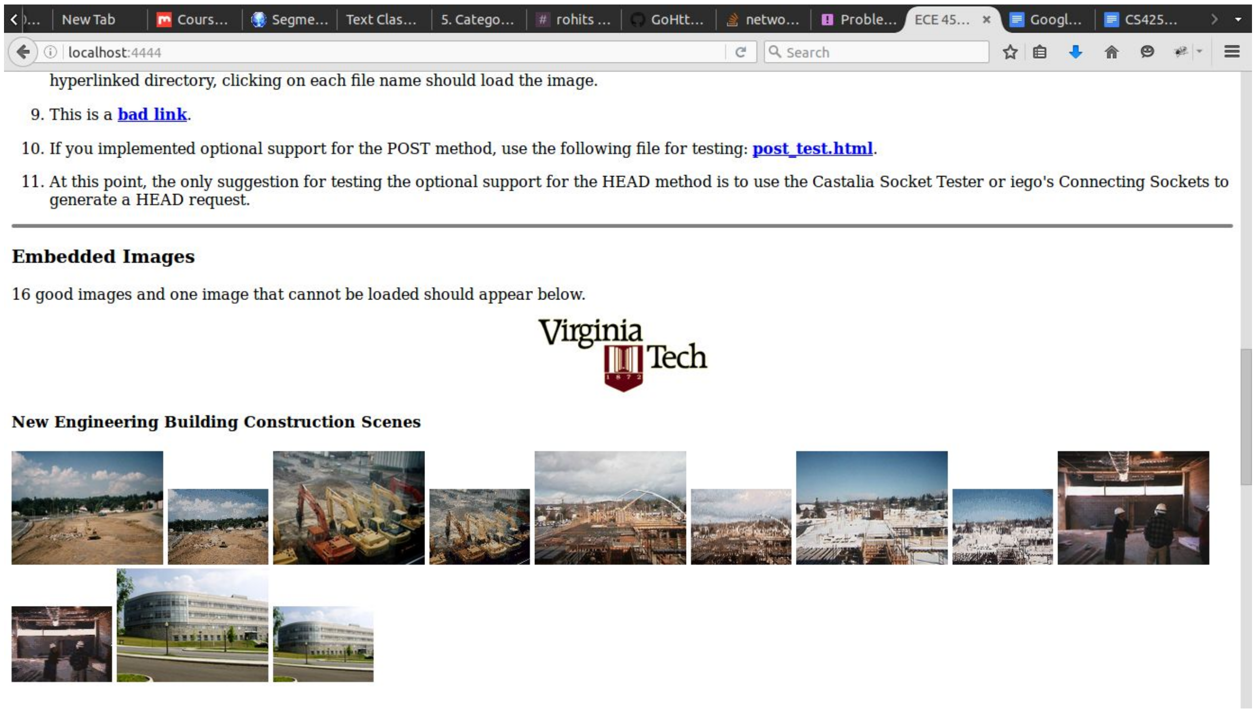
This page and the set of associated files are designed to test your Project 2 HTTP server. Use your browser's options to try and force both HTTP/1.1 and HTTP/1.0 behavior.

Test Files

You should have the following files using the indicated directory structure. *base_directory* is the directory on the server host used as the base directory by your server. For example, the URI `/index.html` should retrieve this file.

```
base_directory

     images
          thm01.gif
          thm02.gif
          thm03.gif
          thm04.gif
          thm05.gif
          thm06.gif
          neb01.jpg
          neb02.jpg
          neb03.jpg
          neb04.jpg
          neb05.jpg
          neb06.jpg
          page.html
          vt.gif
     index.html
```



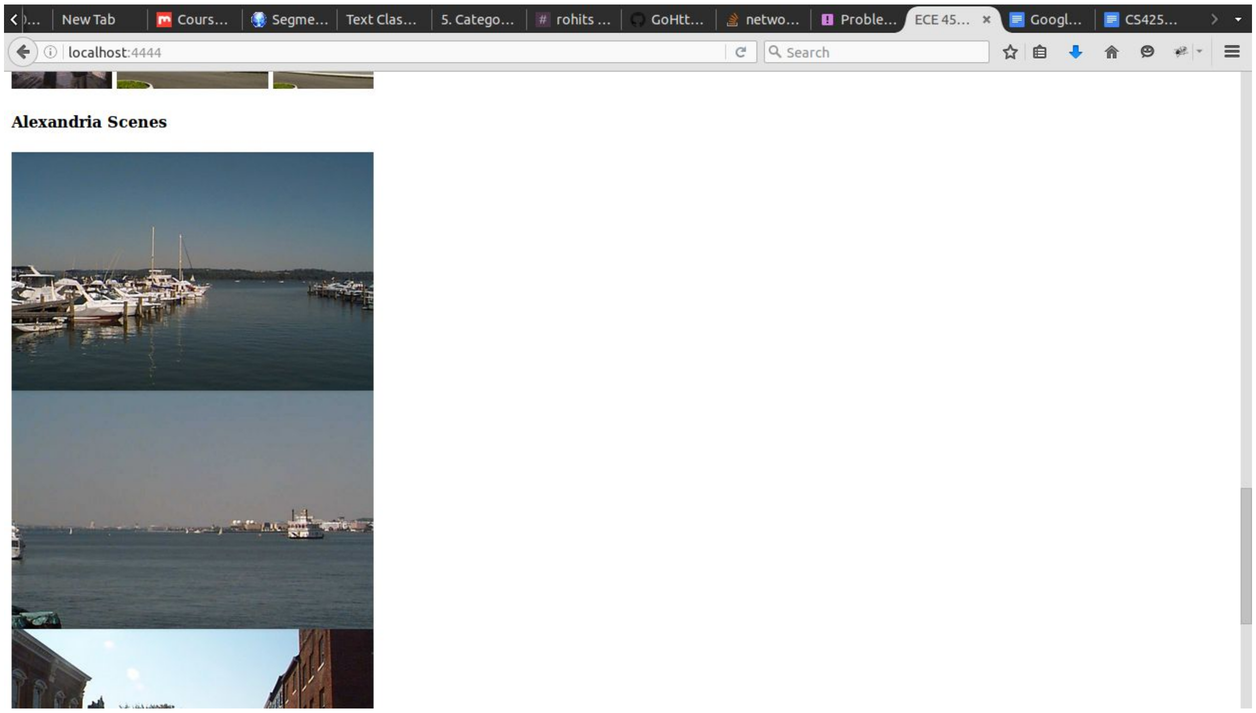
hyperlinked directory, clicking on each file name should load the image.

9. This is a **bad link**.
10. If you implemented optional support for the POST method, use the following file for testing: **post_test.html**.
11. At this point, the only suggestion for testing the optional support for the HEAD method is to use the Castalia Socket Tester or iego's Connecting Sockets to generate a HEAD request.

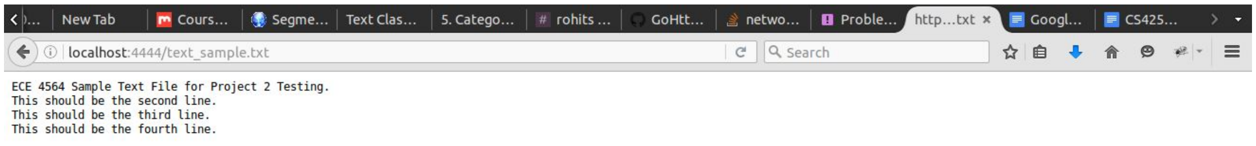
Embedded Images

16 good images and one image that cannot be loaded should appear below.

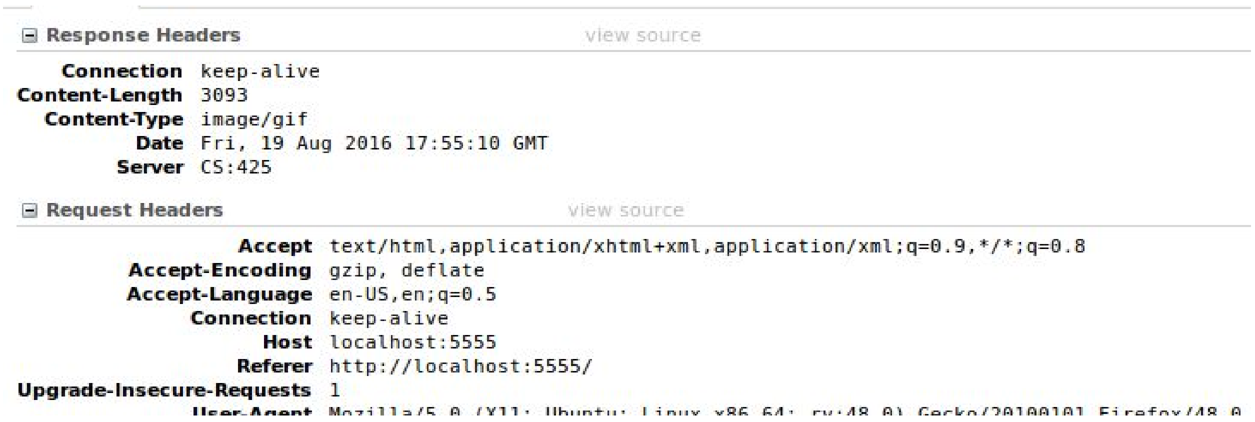
New Engineering Building Construction Scenes



Alexandria Scenes



```
ECE 4564 Sample Text File for Project 2 Testing.
This should be the second line.
This should be the third line.
This should be the fourth line.
```



Response Headers

Connection	keep-alive
Content-Length	3093
Content-Type	image/gif
Date	Fri, 19 Aug 2016 17:55:10 GMT
Server	CS:425

Request Headers

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en-US,en;q=0.5
Connection	keep-alive
Host	localhost:5555
Referer	http://localhost:5555/
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:48.0) Gecko/20100101 Firefox/48.0

Browser Used: Mozilla Firefox, Google-Chrome

APPENDIX

```
#include<iostream>
#include<ctime>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<errno.h>
#include<string>
#include<string.h>
#include<unistd.h>
#include<unordered_map>

#define SUCCESS 1
#define FAILURE 0
#define conf_file "Initial.conf"                    // Configuration File

using namespace std;

int sock                            ;
```

```
unsigned int len                    ;
struct sockaddr_in server, client ;
char file[50]   ;
int status_code ;
char root[200], extra[200]      ;
char Connection_type;
unordered_map <int, const char *>   status_dict ;
unordered_map <string, const char *> media_dict;

void chores(){

        status_dict[200] = " 200 OK\r\n"                        ;
        status_dict[400] = " 400 Bad Request\r\n"       ;
        status_dict[404] = " 404 Not Found\r\n"            ;
        status_dict[500] = " 500 Internal Server Error\r\n"  ;
        status_dict[501] = " 501 Not Implemented\r\n"          ;

        media_dict["html"] = "text/html\r\n"  ;
        media_dict["htm"]  = "text/html\r\n"           ;
        media_dict["txt"]  = "text/plain\r\n"   ;
        media_dict["jpeg"] = "image/jpeg\r\n"       ;
        media_dict["jpg"]  = "image/jpeg\r\n"        ;
        media_dict["gif"]  = "image/gif\r\n"   ;
        media_dict["pdf"]  = "Application/pdf\r\n";
        media_dict["other"]= "Application/octet-stream\r\n";

}

void send_file(FILE *fp, int client, int flength){

        char fbuf[8196] ;
        int bytes = 0   ;
        for (int i = 0; i < flength; i++){
        bytes = fread(fbuf, 1, sizeof(fbuf), fp)       ;
        send(client, fbuf, bytes, 0)     ;
        }

}

char * date_time_header(void){

  char buf[1000];
  memset(buf, '\0', sizeof(buf))        ;
  time_t now = time(0);
  struct tm tm = *gmtime(&now);

  strftime(buf, sizeof (buf), "%a, %d %b %Y %H:%M:%S %Z", &tm);
  return buf;

}

int Send_Headers(int status_code, char * type, int length, int client){

        char  http_version[] = "HTTP/1.1"            ;
        char  server[]       = "Server: CS:425\r\n"           ;
        char  cont_length[]  = "Content-Length: "  ;
        char  cont_type[] = "Content-Type: "       ;
        char  connection[]   = "Connection: "       ;
        char date[]          = "Date: "             ;
        char Whitespace[]        = "\r\n"                 ;
        char buf[20], Headers[200] ;

        char * d = date_time_header()   ;
        memset(buf, '\0', sizeof(buf))  ;
        memset(Headers, '\0', sizeof(Headers));
        sprintf(buf, "%d", length)                    ;
        strcat(Headers, http_version)                 ;
        strcat(Headers, (status_dict[status_code]))        ;
        strcat(Headers, server)                       ;
        strcat(Headers, cont_length)                  ;
        strcat(Headers, buf)                          ;
        strcat(Headers, Whitespace)                   ;
        strcat(Headers, date)                         ;
        strcat(Headers, d)                      ;
        strcat(Headers, Whitespace)                   ;
        strcat(Headers, cont_type)                    ;
        strcat(Headers, media_dict[type])             ;
        strcat(Headers, connection)                   ;
        if (Connection_type == 'k')
        strcat(Headers, "keep-alive\r\n")             ;
        else
        strcat(Headers, "close\r\n")            ;
        strcat(Headers, Whitespace)                   ;
        send(client, Headers, strlen(Headers), 0)          ;
}

char * get_file_type(char * filename){

        memset(file, '\0', sizeof(file))  ;
        int i = 0, j = 0   ;
        for (i = 0; i < strlen(filename); i++)
        if(filename[i] == '.')
        break;
        i++ ;

        if (i > strlen(filename)){
        strcat(file, "else") ;
        return file;
        }

        else{

        memset(file, '\0', sizeof(file))  ;
        while(i < strlen(filename)){
        file[j] = filename[i] ;
        j++    ;
        i++    ;
        }

        if (media_dict.count(file))
        return file;

        strcpy(file, "others")  ;
        return file;
        }
}

int get_file_length(FILE * fp){

        int length = 0;

    fseek(fp, 0, SEEK_END);
    length = ftell(fp);
    rewind(fp);
        return length;

}

void get_one_word(char *out, char *inp, int start_position){

        int position = 0     ;
        for (int i = start_position; i < strlen(inp); i++)
        if (inp[i] != ' '){
        out[position] = inp[i]  ;
        position ++ ;
        }
        else
        break;
}


int process_GET(char * buffer, int client){

        char filename[1024]      ;
        char * file_type    ;
        FILE *fp      ;
        memset(extra, '\0', sizeof(extra)) ;
        memset(filename, '\0', sizeof(filename)) ;

        get_one_word(extra, buffer, 4) ;
        file_type  = get_file_type(extra)   ;

        if (strcmp("else", file_type) == 0)
        sprintf(filename, "%s%s%s", root, extra, "/index.html")  ;
        else
        if ((extra[strlen(extra)-1] == '/'))                              // if the last
character is / then append "index.html" to the filename
        sprintf(filename, "%s%s%s", root, extra, "index.html")  ;
        else
        sprintf(filename, "%s%s", root, extra)   ;

        file_type  = get_file_type(filename)   ;

        fp = fopen(filename, "rb")   ;
        if (fp == NULL){
        printf("Could not open File\n") ;
        status_code = 404;
        return SUCCESS;
        }


        int file_length = get_file_length(fp)          ;

        int header_status = Send_Headers(status_code, file_type,
file_length, client)  ;
        send_file(fp, client, file_length)   ;
        return SUCCESS  ;

}

void connection_type(char * buffer){
        string buf(buffer);
        int index = buf.find("Connection:")   ;

        if (buf[index+strlen("Connection: ")] == 'k')
        Connection_type = 'k'   ;
        else
        Connection_type = 'c'   ;

}

int query_type(char * buffer){

        char inp[512]   ;
        memset(inp, '\0', sizeof(inp))  ;
        printf("%s", buffer)  ;
        get_one_word(inp, buffer, 0) ;

        if (strcmp("GET", inp) == 0)
        return 1      ;
        return 4      ;

}

int process_client(int client){

        int type;
        char buffer[512]   ;
        memset(buffer, '\0', sizeof (buffer))   ;

        int query_length = recv(client, buffer, sizeof(buffer), 0) ;  // Recieving
request from client

        if (query_length <= 0){

        status_code = 400   ;
        return SUCCESS ;

        }

        type = query_type(buffer) ;
        connection_type(buffer) ;

        if (type == 1)
        return process_GET (buffer, client)   ;

        else
        status_code = 501   ;
        return SUCCESS          ;
        }

void Socket(){
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1 ){
/*Establishing a Socket for Server Process*/
        perror("Failed to establish Socket :(\n") ;
        exit(-1)       ;
        }
}

void Bind(int server_port){

        server.sin_family = AF_INET                   ;
        server.sin_port = htons(server_port)       ; // Server Port
        server.sin_addr.s_addr = INADDR_ANY          ;
        bzero(&server.sin_zero, 8)                    ;
        len = sizeof(struct sockaddr_in)              ;

        if ((bind (sock, (struct sockaddr *)&server, len)) == -1){
/*Trying to Bind to the Socket*/
        perror("Failed to bind to Socket :(\n" );
        exit(-1)  ;
        }
}

void listen(){
        if (listen(sock, 5) == -1){                              /*server process
listening on the established Socket*/
        perror("Failed to Listen to client(s)\n") ;
        exit(-1)       ;
        }
}
```

```
void Connecting(){
        int cli = -1   ;
        int pid ;
        while(1){
        if ((cli = accept(sock, (struct sockaddr *)&client, &len)) == -1){
                        exit(-1)       ;
                }
        pid = fork() ;

        if(pid == -1){

        printf("Closing")   ;
        close(cli)  ;
        continue     ;

        }

        if (pid > 0){

        close(cli)  ;
        continue     ;

        }

        if (pid == 0){

        status_code = 200   ;

        int response = process_client(cli)   ;

        if (response == FAILURE)
                close(cli)  ;
```

```
    }



if (Connection_type == 'c')
        close(cli)  ;

else{

        if (status_code == 400){
        char msg[] = "HTTP/1.1 400 Bad Request\r\n"   ;
        send(cli, msg, sizeof(msg), 0)   ;
        continue     ;
        }

        if (status_code == 404){
        char msg[] = "HTTP/1.1 404 Not Found\r\n"   ;
        send(cli, msg, sizeof(msg), 0)   ;
        close(cli)  ;
        close(sock) ;
        continue     ;
        }

        if (status_code == 500){
        char msg[] = "HTTP/1.1 500 Internal Server Error\r\n"   ;
        send(cli, msg, sizeof(msg), 0)   ;
        exit(1) ;
        }

        if (status_code == 501){
        char msg[] = "HTTP/1.1 501 Not Implemented\r\n"   ;
        send(cli, msg, sizeof(msg), 0) ;
        continue     ;
        }
        }
}
```

```
        close(cli)  ;
        break   ;

        }
        close(sock) ;
        }

void init(int server_port){


        Socket()   ;
        Bind(server_port)         ;
        listen()   ;

        while(1)
        Connecting()  ;

}

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

        char port[7];
        FILE *fhandle   ;

        fhandle = fopen(conf_file, "r")         ;
        if (fhandle == NULL){
        printf("Error in openining Configuration File\n")   ;
        exit(1) ;
        }

        if (fscanf(fhandle, "%s %s",extra, root )  != 2)
        printf("Error reading Base Directory from configuration file.\n") ;
        int server_port = atoi(argv[1])         ;
        chores()  ;
        init(server_port);
        return 0      ;

}
```

Testing Results:

- After compiling and running the server on terminal, i tested with Chrome, Firefox browsers and also on Mobile phone
- I opened the server link with it’s port number i.e. localhost:port/ to open index.html
- I opened localhost:4444/ and index.html file came as shown in below image
- I also checked the content of both request and response header and it was same as expected, also shown in below image
-