Report: Project 1

CS425A Computer Networks 19th Aug 2016

Avinash Mohak, 13177 (amohak@iitk.ac.in)

Mandatory features:

1. Supports the GET method
2. Supports persistent connections
3. Provides Content-Length and Content-Type fields in its response
4. Provides appropriate Status-Code and Response-Phrase values in response to errors
5. Server runs continuously until an unrecoverable error occurs
6. Supports concurrent service request from multiple clients

Optional Features Implemented (with their brief discussion):

1. Allowed the server port to be initialized at start up via a command line argument
    - Used **argv** to take port as input
2. Implemented the **HEAD** method
    - When asked for HEAD, same response as GET is sent except the requested file content(message body), i.e., only the status line and response headers are sent.
3. Reply with a hyperlinked directory listing if a directory is the requested resource
    - Library Used to list the files in a directory**: dirent** (included in <dirent.h>)
    - Used **stat** function to determine if the requested path is a file or directory
    - If path is a directory, a temporary html file with links to all the files (using href) in that directory is created, sent to the browser, and the file removed thereafter.
    - **Accessing a directory:** For the base directory, “index.html” is sent while for other directories, the directory listing is sent irrespective of the presence of “index.html” in that directory.

Testing Results: (Browser: **Google Chrome - Version 52.0.2743.116**)

1. GET Request for “/” or “/index.html” or “/index.html/” (Appendix A.1)
2. Response Header for a typical GET Request (Appendix A.2)
3. GET Request reply for “images” directory with hyperlinked files (Appendix A.3)
4. For testing HEAD, curl request was sent from the terminal, and the output shown (Appendix A.4)
5. For testing that POST(or any other request method except GET & HEAD) is not implemented and is handled well. (Appendix A.5)
6. Testing for GET Request for a non-existent file (Appendix A.6)
7. Testing for persistent connections (Appendix A.7)
    - After index.html was requested for the first time, the number of threads created remained constant, even on requesting for various files and directories thereafter.
    - A.7 shows two states of the number of threads, once on serving first request(“index.html”), and the second one after a few (6 in the shown case) requests.

Summary:

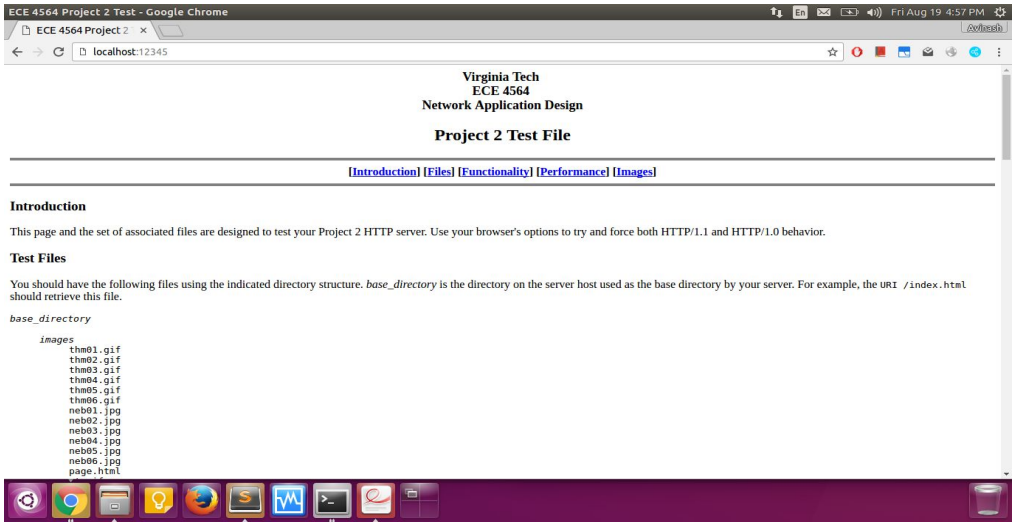
1. All mandatory and above mentioned optional features are working properly to the best of my knowledge.
2. Internal Server Error could not be implemented and tested.
3. When Connection field is given “Close”, the thread executing the request is made to exit after serving the request, but the same could not be tested concretely.
4. Special Characters and spaces in the filenames could not be supported.

Acknowledgement:

For parsing the request from the browser, a modified version of string parsing library (proxy_parse.h), provided with the Project 2, has been used.

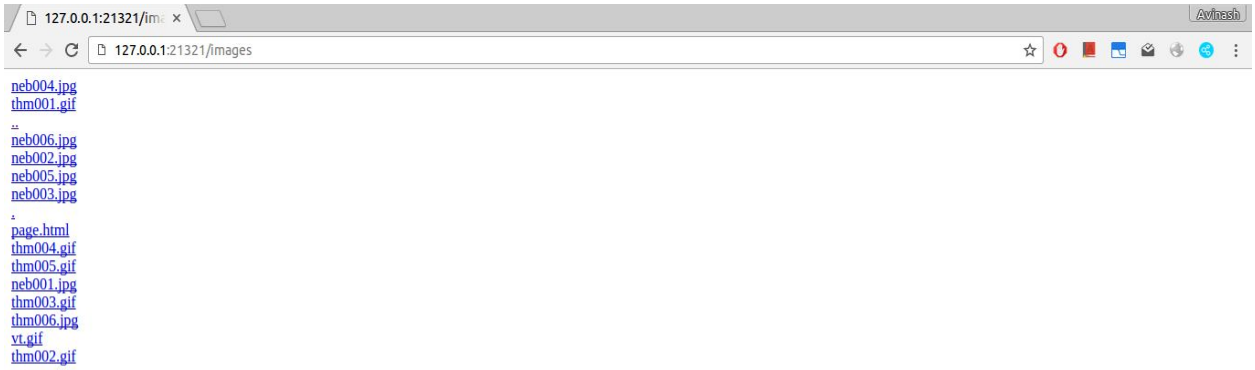
APPENDIX A

1. GET Request

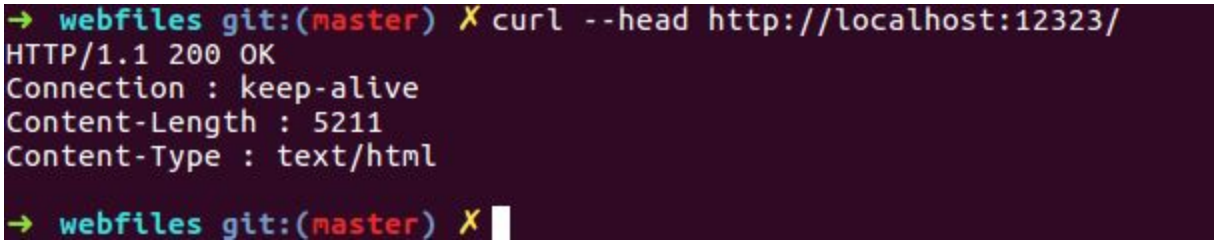


2. Response Header for GET Request

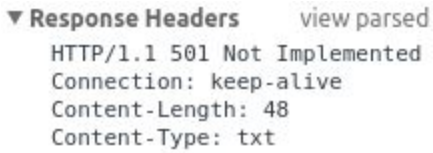
3. Directory Listing



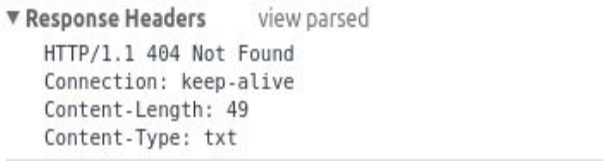
4. Response for HEAD request

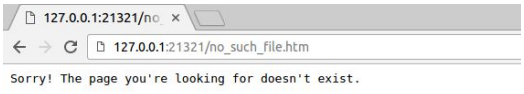


5. POST Request Handling



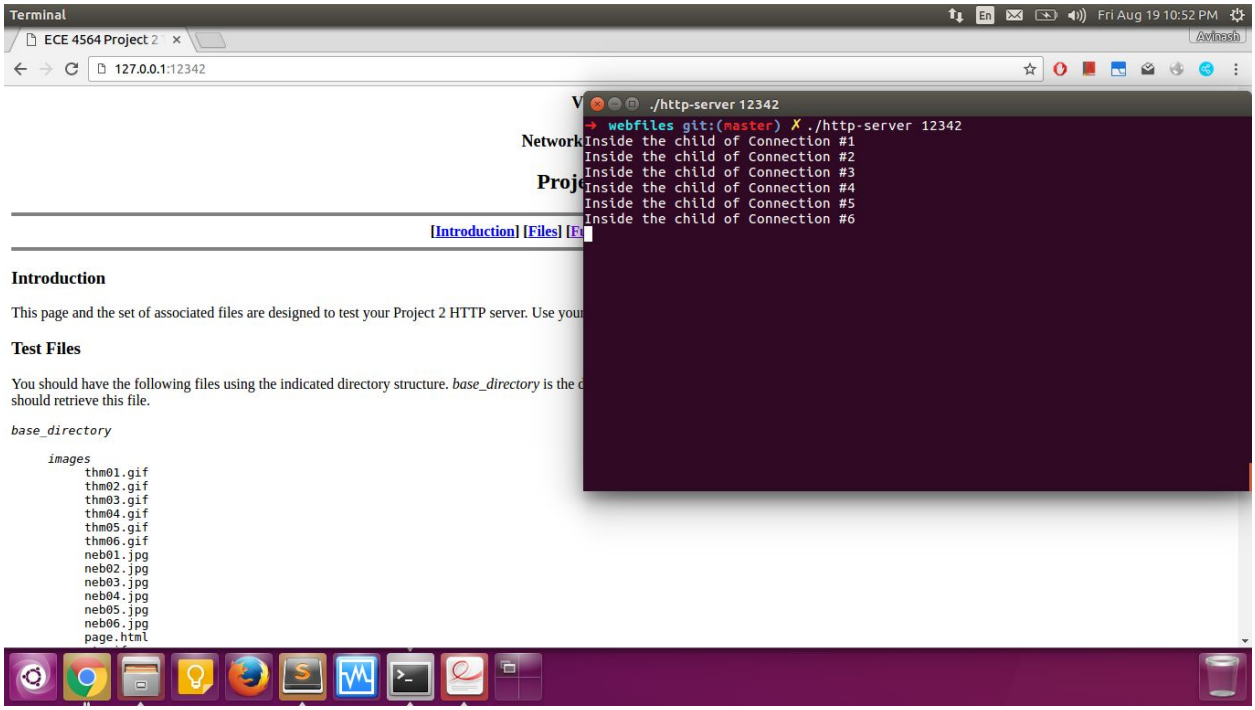
6. Page Not Found Error



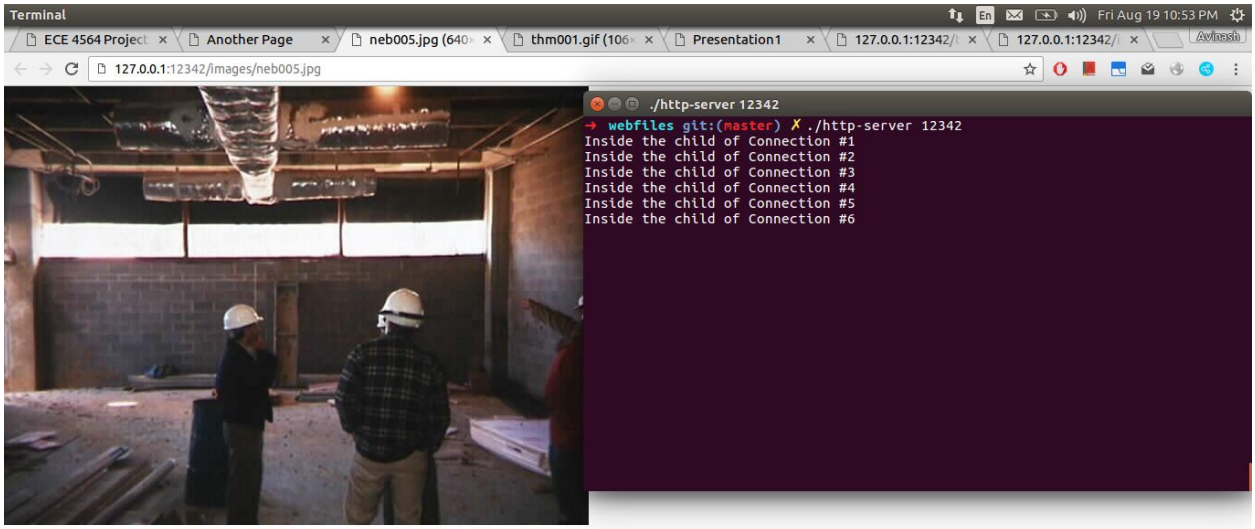


7.

a.



b.



8. **Source Code (server.cpp)**

```
#include <bits/stdc++.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sendfile.h>
#include <unistd.h>
#include <netinet/in.h>
#include <signal.h>
#include <netdb.h>
#include <fcntl.h>

#include "proxy_parse.h"
#include "book_keeping.h"

using namespace std;

#define MAX_PENDING 5
#define MSGRESLEN 8092
#define MAX_BUFFER_SIZE 8092

char BadRequestMessage[] = "Sorry! The page you're looking for doesn't exist.\0";
char *ParsingFailed[2];

void initialise_Message()
{
	ParsingFailed[0] = "Sorry! We didn't get the request properly. You may try to reload
the page.\0";
	ParsingFailed[1] = "Sorry! We do not support the requested facility.\0";
}

string htmlheader = "<HTML><BODY>";
string htmlfooter = "</BODY><HTML>";

int main(int argc, char * argv[])
{
	int s,pid,SERVER_PORT;

	if (argc==2)
		SERVER_PORT = atoi(argv[1]);
	else
	{
		fprintf(stderr, "usage: outfile server_port\n");
		exit(1);
	}
```

```
	initialise_ReasonPhrase();
	initialise_Message();

	struct sockaddr_in sin;
	unsigned int len;
	memset((char *)&sin, 0, sizeof(sin));
	sin.sin_family = AF_INET;
	sin.sin_addr.s_addr = htonl(INADDR_ANY);
	sin.sin_port = htons(SERVER_PORT);

	if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Cannot create the socket");
		exit(1);
	}

	if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0)
	{
		perror("Unable to bind");
		exit(1);
	}

	listen(s, MAX_PENDING);

	int connection_count = 0;

	while(1)
	{
		int new_s;

		len = 10;
		if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
			perror("error in accepting the connection");
			exit(1);
		}

		connection_count++;

		if(fork() == 0)
		{
			cout << "Inside the child of Connection #" << connection_count <<
"\n";
```

```
                        while(1)
                        {

                                char *connection_status;
                                int content_length, retval, type;
                                char *temp_type;
                                struct stat size_info, type_info;
                                char *response_header = (char *)malloc(MAX_BUFFER_SIZE);
                                char *requestmessage = (char *)malloc(MAX_BUFFER_SIZE);

                                if(recv(new_s,requestmessage,MAX_BUFFER_SIZE,0) <= 0){
                                        exit(1);
                                }

                                int l = strlen(requestmessage);
                                requestmessage[l] = '\0';

                                int isHEAD = 0;
                                ParsedRequest *req = ParsedRequest_create();

                                if ((retval = ParsedRequest_parse(req, requestmessage, l)) < 0) {
                                        if(retval == -2)
                                                status_line("HTTP/1.1",501,response_header);
                                        else
                                                status_line("HTTP/1.1",400,response_header);

                                        retval = abs(retval) - 1;

prepare_response("keep-alive",strlen(ParsingFailed[retval]),"txt",response_header);

send_new(new_s,response_header,strlen(response_header));

send_new(new_s,ParsingFailed[retval],strlen(ParsingFailed[retval]));
                                        continue;
                                }

                                if(strcmp(req->method,"HEAD")==0) isHEAD = 1;

                                ParsedHeader header;
                                for(int i=0;i<req->headersused;i++)
                                {
                                        header = *(req->headers+i);
                                        if(strcmp(LowerCase(header.key),"connection")==0)
```

```
                                        {
                                                connection_status = (char
*)malloc(strlen(header.value));
                                                strcpy(connection_status,header.value);
                                        }
                                }

                                if(connection_status == NULL)
                                        connection_status = "keep-alive";


                                if(req->path[strlen(req->path)-1]=='/')
                        // trim the terminal slash character
                                        req->path[strlen(req->path)-1] = '\0';

                                type = stat(req->path,&type_info);

                                int file_descriptor;

                                int f = open(req->path,O_RDONLY);

                                if(type != 0)
                                {
                                        status_line(req->version,404,response_header);

prepare_response(connection_status,strlen(BadRequestMessage),"txt",response_header);

send_new(new_s,response_header,strlen(response_header));
                                        if(!isHEAD)

send_new(new_s,BadRequestMessage,strlen(BadRequestMessage));
                                }

                                else
                                {
                                        status_line(req->version,200,response_header);

                                        if(S_ISREG(type_info.st_mode))
                                        {
                                                file_descriptor = f;
                                                temp_type = extract_type(req->path);
                                        }
```

```
                                        if(S_ISDIR(type_info.st_mode))
                                        {
                                                DIR * dir = opendir(req->path);
                                                FILE *fp = fopen("directory_list.html","w");

                                                if(fp == NULL)
                                                        exit(1);

                                                fprintf(fp,"%s",htmlheader.c_str());
                                                list_dir_file(dir,fp,req->path);
                                                fprintf(fp,"%s",htmlfooter.c_str());
                                                fclose(fp);

                                                file_descriptor =
open("directory_list.html",O_RDONLY);
                                                temp_type = "html";
                                        }

                                        if(fstat(file_descriptor,&size_info) < 0)
                                        {
                                                perror("Error in accessing file stats");
                                                exit(1);
                                        }

                                        content_length =  size_info.st_size;
                                        string content_type = MediaType(temp_type);


prepare_response(connection_status,content_length,content_type,response_header);


send_new(new_s,response_header,strlen(response_header));

                                        if(!isHEAD)

sendfile_new(new_s,file_descriptor,content_length);

                                        if(S_ISDIR(type_info.st_mode))
                                        {
                                                int ret = remove("directory_list.html");
                                                if(ret != 0)
                                                        exit(1);
                                        }
```

```
                                        if(strcmp(connection_status,"close")==0)
                                        {
                                                close(new_s);
                                                break;
                                        }
                                }
                        }
                }
        }
        return 0;
}
```