COMPUTER NETWORKS(CS425A)

PROJECT 1:HTTP SERVER

19th August, 2016

KRITI JOSHI - 13358

kritij@iitk.ac.in

IMPLEMENTED OPTIONS:

1) Allow the server port to be initialized at start up, for example via a command line argument or an initialization file.
2) Allow the document base directory to be initialized at start up, for example via a command line argument or an initialization file.
3) Reply with a directory listing if a directory is the requested resource. Reply with a hyperlinked directory listing if a directory is the requested resource.

Above three are implemented in addition to the mandatory functions.

EXPLANATIONS:

1) Server port number:

Takes input from command line. Basically the server executable has to be run in the given format: **./server <port-number>**

Prompts the user in case port-number is not provided.

2) Document Base Directory:

Prompts user to enter the base Directory.

```
Enter Base Directory:webfiles
Base Directory is:webfiles
```

3) Hyper linked directory listing:

With the help of **<dirent.h>** library, a list of hyperlinked filenames is displayed. This listing appears only if **index.html** file is not present in the given directory.

TESTING:

Tested On : Google Chrome

Host ; 127.0.0.1:9090

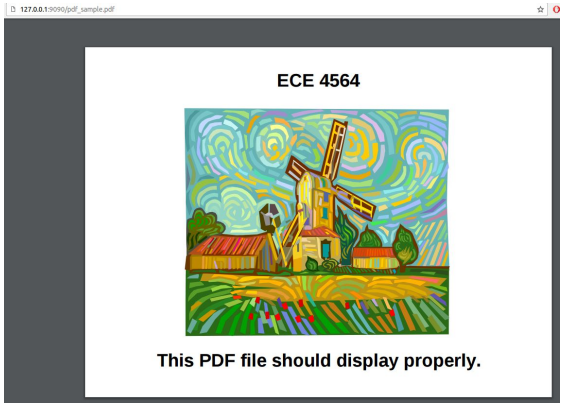
All the given test cases (Testing Functionality in index.html) were properly executed.

1) The following three links should all load this page: **index.html**, **/index.html**, and **/**. (All three options opened the main html page)
2) The following link should load another HTML page: **page.html**. On clicking the test page hyperlink, the main html file opened again.

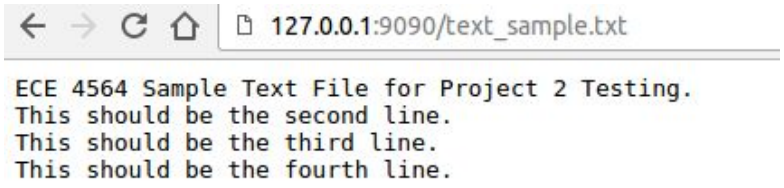
3) The following link should load a JPEG file (which happens to be NEB 261 in its infancy): **neb005.jpg**.

4) The following link should load a GIF file: **thm001.gif**.

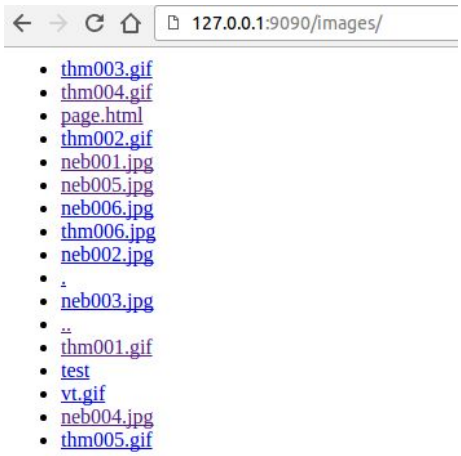
5) The following link should load a PDF file: **pdf_sample.pdf**.



6) The following link should load a text file: text_sample.txt



7) Hyper-linked directory listing.



8) This is a bad link.

APPENDIX:

server.cpp

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <bits/stdc++.h>
#include <dirent.h>
#include <netinet/tcp.h>
// To check whether file or directory
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
// Create thread
#include <pthread.h>

#define BUFFER_SIZE 4096
#define LARGE 150000
#define SMALL 30
#define VERY_SMALL 10
#define NUM_THREADS 100
```

```
using namespace std;

struct REQUEST{
    char* method;
    char* connection;
    char* path;
    char* version;
    char* host;
};

struct RESPONSE{
    int statusCode;
    char* responseCode;
    int contentLength;
    char* contentType;
};

char baseDir[SMALL]="webfiles";

// From stack overflow
//
http://stackoverflow.com/questions/4553012/checking-if-a-file-is-a-directory-or-just-a-file
/*
    Input : path
    Output: True if the path points to a file
*/
int is_regular_file(char path[SMALL]){
```

```
    struct stat path_stat;
    stat(path, &path_stat);
    return S_ISREG(path_stat.st_mode);
}

/*
    Input : path
    Output: True if the path points to a directory
*/
int isDirectory(char path[SMALL]) {
   struct stat statbuf;
   if (stat(path, &statbuf) != 0)
       return 0;
   return S_ISDIR(statbuf.st_mode);
}

/*
    Function: Handles client's request
*/
void* start_routine(void* cfd){
    int clientFileDescriptor = *((int*)(&cfd));
    char input[BUFFER_SIZE];
    bzero(input,BUFFER_SIZE);
    while(true){
        char* saveptr;
        // Receive request till we hit [CRLF][CRLF]
        bzero(input,BUFFER_SIZE);
```

```
int recData = recv(clientFileDescriptor,input,BUFFER_SIZE,0);
if(recData<=0){
    cout<<"Client Served\n";
    pthread_exit (NULL);
}
cout<<recData<<endl;
char* endLoc = strstr(input,"\r\n\r\n");
while(endLoc==NULL){
    recData = recv(clientFileDescriptor,input+recData,BUFFER_SIZE,0);
    endLoc = strstr(input,"\r\n\r\n");
}

// Divide string into tokens separated by whitespaces
REQUEST req;

req.method = strtok_r(input, " ",&saveptr);
// Prepend baseDir to path to get location of file
req.path = strtok_r(NULL," ",&saveptr);
char fullPath[SMALL]="";
strcat(fullPath,baseDir);
strcat(fullPath,req.path);

// Set version
req.version = strtok_r(NULL, "\r",&saveptr);
cout <<"version:"<< req.version <<endl;

// Set Connection
```

```
char* temp = strstr(saveptr,"Connection: ")+strlen("Connection: ");
if(temp==NULL){
    cout<<"GET doesn't contain Connection type\n";
    pthread_exit (NULL);
}
int lenConn = strstr(temp,"\r\n")-temp;
req.connection = (char*)malloc(lenConn);
strncpy(req.connection,temp,lenConn);

// Create response
RESPONSE resp;
char msgBody[LARGE];
bzero(msgBody,LARGE);

// If file is asked
if(is_regular_file(fullPath)){
    // set status code and response code
    resp.statusCode = 200;
    resp.responseCode = (char*)malloc(sizeof("OK"));
    strcpy(resp.responseCode,"OK");

    // content-type
    char* extension = strrchr(req.path,'.')+1;
    FILE* fp;
    resp.contentType = new char(SMALL);
    if(!strncmp(extension,"htm",3)){
        strcpy(resp.contentType,"text/html");
```

```
    fp = fopen(fullPath,"r");
}
else if(!strncmp(extension,"txt",3)){
    strcpy(resp.contentType,"text/plain");
    fp = fopen(fullPath,"r");
}
else if(!strncmp(extension,"jpg",3)){
    strcpy(resp.contentType,"image/jpeg");
    fp = fopen(fullPath,"rb");
}
else if(!strncmp(extension,"gif",3)){
    strcpy(resp.contentType,"image/gif");
    fp = fopen(fullPath,"rb");
}
else if(!strncmp(extension,"pdf",3)){
    strcpy(resp.contentType,"Application/pdf");
    fp = fopen(fullPath,"r");
}
else cout<<"Something fishy o.O \n";

// file size
fseek(fp, 0L, SEEK_END);
resp.contentLength = ftell(fp);
fseek(fp, 0L, SEEK_SET);

if(resp.contentLength > LARGE){
    cout <<"file larger than LARGE\n";
```

```
        }

        int bytesRead = fread(msgBody, 1,resp.contentLength, fp);
    } // If directory is asked
    else if(isDirectory(fullPath)){
        // status and response code
        resp.statusCode = 200;
        resp.responseCode = (char*)malloc(sizeof("OK"));
        strcpy(resp.responseCode,"OK");

        char indexPath[LARGE];
        bzero(indexPath,LARGE);
        if(fullPath[strlen(fullPath)-1]=='/')
            fullPath[strlen(fullPath)-1]='\0';
        sprintf(indexPath,"%s/index.html",fullPath);
        cout<<"indexPath:"<<indexPath<<endl;
        FILE* fp = fopen(indexPath,"rb");

        // index.html exists
        if(fp!=NULL){
            resp.contentType = (char*)malloc(sizeof("text/html"));
            strcpy(resp.contentType,"text/html");

            fseek(fp, 0L, SEEK_END);
            resp.contentLength = ftell(fp);
            fseek(fp, 0L, SEEK_SET);
            cout<<"Length actual:"<<resp.contentLength<<endl;
```

```
            if(resp.contentLength > LARGE){
               cout <<"file larger than LARGE\n";
            }

            int bytesRead = fread(msgBody, 1,resp.contentLength, fp);
            cout<<"Read Bytes:"<<bytesRead<<endl;
         } // If index.html doesn't exist
         else{
            // return directory listing
            // reference:
http://stackoverflow.com/questions/612097/how-can-i-get-the-list-of-files-in-a-directory-using-c-or-c

            // Directory listing
            DIR *dir;
            struct dirent *ent;
            if((dir = opendir (fullPath)) != NULL){
               /* print all the files and directories within directory */
               int numFiles =0;
               //char msgBody[BUFFER_SIZE];
               sprintf(msgBody,"<HTML><HEAD><TITLE>Directory
Listing</TITLE></HEAD></HTML><BODY><ul>");
               while((ent = readdir (dir)) != NULL){
                  if(ent->d_name!="." && ent->d_name!=".."){
                     char link[BUFFER_SIZE];
                     bzero(link,BUFFER_SIZE);
                     sprintf(link,"<li><a
href=\"%s/%s\">%s</a></li>\n",req.path,ent->d_name,ent->d_name);
```

```
                cout<<"link:"<<link<<endl;
                strcat(msgBody,link);
              }
            }
            strcat(msgBody,"</ul></BODY></HTML>");
            cout<<"MsgBody:"<<msgBody<<endl;
            resp.contentLength = strlen(msgBody);
            resp.contentType = (char*)malloc(sizeof("text/html"));
            strcpy(resp.contentType,"text/html");
        }else{
            goto label;
        }
    }

}// If file doesn't exist
else{
    label:
    cout<<"Error\n";

    // status code and response code
    resp.statusCode = 404;
    resp.responseCode = (char*)malloc(sizeof("Not Found"));
    strcpy(resp.responseCode,"Not Found");

    // content type
    char* extension = strrchr(req.path,'.')+1;
    FILE* fp;
```

```
resp.contentType = new char(SMALL);
if(!strncmp(extension,"htm",3)){
   strcpy(resp.contentType,"text/html");
   fp = fopen(fullPath,"r");
}
else if(!strncmp(extension,"txt",3)){
   strcpy(resp.contentType,"text/plain");
   fp = fopen(fullPath,"rb");
}
else if(!strncmp(extension,"jpg",3)){
   strcpy(resp.contentType,"image/jpeg");
   fp = fopen(fullPath,"rb");
}
else if(!strncmp(extension,"gif",3)){
   strcpy(resp.contentType,"image/gif");
   fp = fopen(fullPath,"rb");
}
else if(!strncmp(extension,"pdf",3)){
   strcpy(resp.contentType,"Application/pdf");
   fp = fopen(fullPath,"rb");
}
else cout<<"Something fishy o.O \n";
cout<<"type:"<<resp.contentType<<endl;

// create reply
char reply[LARGE];
bzero(reply,LARGE);
```

```
        char msgBody[BUFFER_SIZE];
        sprintf(msgBody,"<HEAD><TITLE>404 Not Found</TITLE></HEAD>"
                "<BODY><H1>404 Not Found</H1>"
                "The requested URL <strong>%s</strong> was not found on this
server.</BODY>",req.path);
        // Current time
        // Ref: http://www.tutorialspoint.com/cplusplus/cpp_date_time.htm
        time_t now = time(0);
        char* dt = ctime(&now);
        resp.contentLength = strlen(msgBody);
        // resp.contentLength = 0;
        sprintf(reply, "%s %d %s\r\nContent-Type: %s\r\n\r\n",
           req.version,resp.statusCode,resp.responseCode,resp.contentType);


        //send reply
        int len = strlen(reply);
        int dataSent = send(clientFileDescriptor,reply,len,0);
        cout<<reply<<endl;
        while(dataSent<len){
           cout<<"\n\nOnly "<<dataSent<<" bytes sent till now instead of "<<len<<"
bytes\n\n";
           dataSent += send(clientFileDescriptor,reply+dataSent,BUFFER_SIZE,0);
        }


        // Send message body
        dataSent = send(clientFileDescriptor,msgBody,resp.contentLength,0);
        while(dataSent<resp.contentLength){
           cout<<"\n\nOnly "<<dataSent<<" bytes sent till now instead of
"<<resp.contentLength<<" bytes\n\n";
```

```
            dataSent += send(clientFileDescriptor,msgBody+dataSent,BUFFER_SIZE,0);
        }
        cout<<"Data sent successfully:"<<dataSent<<"\n";
        continue;
        pthread_exit (NULL);
    }

    // create header
    char header[LARGE];
    bzero(header,LARGE);
    sprintf(header, "%s %d %s\r\nContent-Length: %d\r\nContent-Type: %s\r\n\r\n",

req.version,resp.statusCode,resp.responseCode,resp.contentLength,resp.contentType);
    cout<<"Header:"<<header<<endl;

    //send header
    int len = strlen(header);
    int dataSent = send(clientFileDescriptor,header,len,0);
    // cout<<dataSent<<endl;

    // Send message body
    dataSent = send(clientFileDescriptor,msgBody,BUFFER_SIZE,0);
    while(dataSent<resp.contentLength){
        cout<<"\n\nOnly "<<dataSent<<" bytes sent till now instead of
"<<resp.contentLength<<" bytes\n\n";
        dataSent += send(clientFileDescriptor,msgBody+dataSent,BUFFER_SIZE,0);
    }
    cout<<"Data sent successfully:"<<dataSent<<"\n";
```

```
        if(!strcmp(req.connection,"Close")){
            cout<<"Close request\n";
            pthread_exit(NULL);
        }
    }
    pthread_exit (NULL);
}

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

    if(argc!=2){
        cout<<"Please enter port-number\n";
        return 1;
    }

    // Decide base directory
    cout<<"Enter Base Directory:";
    scanf("%s",baseDir);
    cout<<"Base Directory is:"<<baseDir<<endl;

    // Create Socket
    int socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
    int flag = 1;
    if(socketFileDescriptor<0){
        cout<<"Couldn't create socket\n";
        return 1;
    }
```

```
    //server Address information
    struct sockaddr_in serverAddress, clientAddress;
    serverAddress.sin_family = AF_INET;
    // serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1");         //binds only to localhost
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);              //binds to all available
interfaces
    serverAddress.sin_port = htons(atoi(argv[1]));


    //Assign socket address to declared socket
    int fails=bind(socketFileDescriptor, (struct sockaddr*)&serverAddress,
sizeof(serverAddress));
    if(fails){
        fprintf(stderr,"Couldn't bind to the port: %d\n",atoi(argv[1]));
        return 1;
    }


    //Start listening on the port, maximum allowed clients is 5
    fails=listen(socketFileDescriptor,5);
    if(!fails){
        cout<<"Listening...\n";
    }


    // handle multiple clients
    while(true){
        long clientFileDescriptor = accept(socketFileDescriptor,NULL,NULL);
        pthread_t* thread=(pthread_t*)malloc(sizeof(pthread_t));
        // assign a thread to a client
```

```
        int error = pthread_create (thread, NULL, start_routine, (void*)clientFileDescriptor);
        if (error){
            cout <<"Thread coudn't be created"<< error << endl;
            exit(-1);
        }
    }
    pthread_exit(NULL);
    return 0;
}
```