#include "server.h"

struct Request{

    char\* type;

    char\* path;

    char\* protocol;

    char\* fileEnding;

}client\_request;

int main(int argc, char \*argv[]){

    //Create a socket

    int err\_flag=0;

    int port;

    int verbose=0;

    if(argc>1){

        for(int i=0; i<argc;i++){

            if(strcmp(argv[i],"-d")==0){

                verbose=1;

            }

            if(strcmp(argv[i],"-p")==0){

                port = atoi(argv[i+1]);

            }

        }

    }

    int socket\_desc;

    int client\_sock, c, read\_size;

    char client\_message[1000];

    struct sockaddr\_in server, client;

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

    if(verbose==1 & socket\_desc > -1){

        printf("Socket created\n");

    }

    else if(socket\_desc == -1){

        printf("Could not create a socket\n");

        return 1;

    }

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

    server.sin\_family = AF\_INET;

    if(port>0){

        server.sin\_port = htons(port);

    }

    else{

        server.sin\_port = htons(8888);

    }

    //bind socket

    if( bind(socket\_desc,(struct sockaddr \*)&server , sizeof(server)) < 0){

        perror("bind failed");

        return 1;

    }

     else if(verbose==1){

        printf ("Bind successful\n");

        if(port>0){

        printf ("Listening at port: %d\n",port);

        }

        else{

            printf("Listening at port: 8888\n");

        }

     }

    //listen to socket

    listen(socket\_desc, 3);

        //Accept incoming

    puts("Waiting for incoming connections ... .. . ..");

    c=sizeof(struct sockaddr\_in);

        //connect

    client\_sock=accept(socket\_desc, (struct sockaddr\*)&client, (socklen\_t\*)&c);

     if (client\_sock < 0){

             perror("accept failed");

             return 1;

    }

    puts("Connection accepted");

    write(client\_sock, "Disconnect by typing : '!Q'\n",28);

    //main loop to handle client requests

    MAIN:

    while((read\_size = recv(client\_sock, client\_message, sizeof(client\_message),0)) > 0){

    //divide incoming string into tokens

        char\* token = strtok(client\_message, " ");

        int i = 1;

        while (token != NULL) {

            chop\_newLine(token);

            if ((strcmp(token, "\0")==0)||(strcmp(token,"\n")==0)){

                break;

            }

            if((strcmp(token, "GET") != 0 && strcmp(token, "POST") != 0  && strcmp(token, "HEAD") != 0 && strcmp(token,"!Q")!=0)&&(i==1)){

                send\_error(client\_sock, NOT\_IMPLEMENTED);

                err\_flag=1;

                break;

            }

            else if(i==1){

                chop\_newLine(token);

                puts(token);

                if(strcmp(token,"!Q")==0){

                    exit(0);

                }

                client\_request.type=token;

            }

            else if(i == 2){

                if(token==NULL){

                    send\_error(client\_sock, BAD\_REQUEST);

                    err\_flag=1;

                    break;

                }

                client\_request.path = token;

            }

            else if(i == 3){

                if (token ==NULL){

                    send\_error(client\_sock, BAD\_REQUEST);

                    err\_flag=1;

                    break;

                }

                client\_request.protocol = token;

            }

            token = strtok(NULL, " ");

            ++i;

        }

        if(err\_flag==1){

            err\_flag=0;

            break;

        }

        if(client\_request.protocol == NULL && client\_request.type==NULL && client\_request.path==NULL){

            goto MAIN;

        }

        if (client\_request.protocol != NULL && client\_request.type!=NULL && client\_request.path!=NULL){

            chop\_newLine(client\_request.protocol);

            char temp[sizeof(client\_request.path)];

            char\*fileEnding = malloc(sizeof(client\_request.path));

            memcpy(fileEnding,client\_request.path,sizeof(client\_request.path)\*4);

            fileEnding=strtok(fileEnding,".");

            fileEnding=strtok(NULL,".");

            if((check\_forbidden(client\_request.path)==1)){

                send\_error(client\_sock, FORBIDDEN);

                break;

            }

            if((check\_config(client\_request.protocol)==1) & (check\_config(fileEnding)==1)){

                process\_request(client\_sock, client\_request.type, client\_request.path);

            }

            else{

                send\_error(client\_sock,BAD\_REQUEST);

                break;

            }

        }

        else{

            send\_error(client\_sock,BAD\_REQUEST);

            break;

        }

    }

    if(read\_size==0){

        puts("Client disconnected");

    }

    else if(read\_size==-1){

        perror("receive failed");

    }

    exit (0);

}