#include "server.h"

void chop\_newLine(char \*s) {

    while(\*s && \*s != '\n' && \*s != '\r') s++;

    \*s = 0;

}

int check\_config(const char \* str){

    FILE \* fp;

    int isFound=0;

    fp=fopen("conf.cfg","r");

    char line[1000];

    char token[50];

    if(fp!=NULL){

        while (fscanf(fp, "%s", token) != EOF) {

            if(strcmp(token,str)==0){

                    isFound=1;

                    break;

            }

        }

    }

    else{

        perror("Missing config file");

    }

    fclose(fp);

    return isFound;

}

int check\_forbidden(const char \* str){

    FILE \* fp;

    int isFound=0;

    fp=fopen("forbidden.cfg","r");

    char line[1000];

    char token[50];

    if(fp!=NULL){

        while (fscanf(fp, "%s", token) != EOF) {

            if(strcmp(token,str)==0){

                    isFound=1;

                    break;

            }

        }

    }

    else{

        perror("Missing forbidden file");

    }

    fclose(fp);

    return isFound;

}

void slice\_str(const char \* str, char \* buffer, int start, int end)

{

    int j = 0;

    for ( int i = start; i < end; i++, j++ ) {

        buffer[j] = str[i];

    }

}

/\*

Helper function needed for dynamic memory allocation

Input: 123 (int - a number)

Output: 3 (int - number of characters in number)

\*/

int content\_chars\_length(int n){

    int count;

    while (n != 0) {

             n /= 10;

             ++count;

        }

    return count;

}

/\*

Headers to be displayed to client when they receive a response from server

\*/

void write\_headers(int client\_sock, int message\_length){

        struct tm strtime;

        time\_t timeoftheday;

        struct tm \*loc\_time;

        timeoftheday=time(NULL);

        loc\_time=localtime(&timeoftheday);

        char message [21];

        sprintf(message, "\nContent-Length: %d", message\_length);

        write (client\_sock,"\nHost-Name: 10.17.175.206",25);

        write(client\_sock,message,22);

        write(client\_sock,"\n",2);

        write(client\_sock,asctime(loc\_time),strlen(asctime(loc\_time)));

        write(client\_sock,"Content-type: txt/html\n",23);

        write(client\_sock,"\n",2);

}

/\*

INPUT: char\* file contains name of a file that's being read as a part of client's request.

char\* file always starts with '/' character.

OUTPUT: char\* get\_filename returns the name of file without '/' character.

\*/

char\* get\_filename(char\* file){

    if(file[0] == '/'){

        file++;

    }

    return file;

}

int send\_get(int client\_sock, char\* file){

    FILE\* f;

    char\* temp\_filename = get\_filename(file);

    int status=OK;

    if((f = fopen(temp\_filename,"r")) == NULL){

        status=NOT\_FOUND;

    }

    char S[1000];

    if(status == OK){

        while(fgets(S, 1000, f) != NULL){

            fflush(stdout);

        }

    send\_status(client\_sock, status);

    write\_headers(client\_sock, strlen(S));

    write(client\_sock, S, strlen(S));

    fclose(f);

    return(0);

    }

    else{

        send\_status(client\_sock, status);

        write\_headers(client\_sock, strlen(S));

        return(1);

    }

}

int send\_head(int client\_sock, char\* file){

    FILE\* f;

    char\* temp\_filename = get\_filename(file);

    int status=OK;

    if((f = fopen(temp\_filename,"r")) == NULL){

        status=NOT\_FOUND;

    }

    char S[1000];

    if(status == OK){

        while(fgets(S, 1000, f) != NULL){

            fflush(stdout);

        }

        send\_status(client\_sock, status);

        write\_headers(client\_sock, strlen(S));

        return(0);

        fclose(f);

    }

    else{

    send\_status(client\_sock, status);

    write\_headers(client\_sock, strlen(S));

    return(1);

    }

}

int send\_post(int client\_sock, char\* file){

    FILE \*fp;

    int status = CREATED;

    if((fp = fopen(get\_filename(file),"w+")) == NULL){

        status = NOT\_FOUND;

    }

    int hasHitEnter=0;

    char client\_message2[1000];

    char client\_message3[1000];

    int read\_size;

    if(fp != NULL){

    while((read\_size = recv(client\_sock, client\_message2, sizeof(client\_message2),0 ))>0){

                char \*token2=strtok((client\_message2),":");

                char\* secondLine=token2;

                if(strcmp(secondLine,"Content-Length")==0){

                    token2=strtok(NULL,":");

                    int contentLength=atoi(token2);

                    int chars\_written=0;

                    char toWrite[contentLength];

                    while((read\_size = recv(client\_sock, client\_message3, sizeof(client\_message3),0 ))>0){

                            chop\_newLine(client\_message3);

                            if(strcmp(client\_message3,"\0")==0 || strcmp(client\_message3,"\n")==0){

                                if (hasHitEnter==1){

                                    //add last string then print

                                    fwrite(toWrite,sizeof(char),sizeof(toWrite),fp);

                                    fclose(fp);

                                    send\_status(client\_sock, status);

                                    write\_headers(client\_sock, contentLength);

                                    return(0);

                                }

                                else{

                                    toWrite[chars\_written++]='\0';

                                    hasHitEnter=1;

                                }

                            }

                            else{

                                int i=0;

                                while(i<=strlen(client\_message3)&&chars\_written<=contentLength){

                                    toWrite[chars\_written]=client\_message3[i];

                                    chars\_written++;

                                    i++;

                                }

                            }

                            // char toWrite[contentLength];

                            // slice\_str(client\_message3,toWrite,0,contentLength);

                            // fwrite(toWrite,sizeof(char),sizeof(toWrite),fp);

                            // fclose(fp);

                            // send\_status(client\_sock, status);

                            // write\_headers(client\_sock, contentLength);

                            // return(0);

                        }

                    }

                    status=BAD\_REQUEST;

                    send\_error(client\_sock,status);

                    return(1);

                }

    }

}

/\*

GET request implementation

to-do: HEAD and POST

\*/

int process\_request(int client\_sock, char\* type, char\* file){

    if(strcmp(type, "GET") == 0){

        if(send\_get(client\_sock, file)==0){

            printf("Success");

            return (0);

        }

        else{

            printf("FAIL");

            return (1);

        }

    }

    if(strcmp(type, "HEAD") == 0){

        if(send\_head(client\_sock, file)==0){

            return (0);

            printf("Success");

        }

        else{

            printf("FAIL");

            return (1);

        }

    }

    if(strcmp(type, "POST") == 0){

        if(send\_post(client\_sock, file)==0){

            return (0);

            printf("Success");

        }

        else{

            printf("FAIL");

            return (1);

        }

    }

    return(1);

}

void send\_status(int client\_sock, int status){

    char status\_code\_message[20];

    switch(status){

        case 200:

            sprintf(status\_code\_message, "\nStatus Code: %d", OK);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        case 201:

            sprintf(status\_code\_message, "\nStatus Code: %d", CREATED);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        case 403:

            sprintf(status\_code\_message, "\nStatus Code: %d", FORBIDDEN);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        case 400:

            sprintf(status\_code\_message, "\nStatus Code: %d", BAD\_REQUEST);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        case 404:

            sprintf(status\_code\_message, "\nStatus Code: %d", NOT\_FOUND);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        case 501:

            sprintf(status\_code\_message, "\nStatus Code: %d", NOT\_IMPLEMENTED);

            write(client\_sock, status\_code\_message, strlen(status\_code\_message));

            break;

        default:

            write(client\_sock, "\nStatus Code: 500", 19);

            break;

    }

}

/\*

Function that sends a default headers in response to unsuccessful requests

\*/

void send\_error(int client\_sock, int status\_code){

    struct tm strtime;

    time\_t timeoftheday;

    struct tm \*loc\_time;

    timeoftheday=time(NULL);

    loc\_time=localtime(&timeoftheday);

    send\_status(client\_sock, status\_code);

    write(client\_sock, "\nContent-Length: 0", 19);

    write(client\_sock,"\nHost-Name: 10.17.175.206",25);

    write(client\_sock,"\n",2);

    write(client\_sock,asctime(loc\_time),strlen(asctime(loc\_time)));

    write(client\_sock,"\n",2);

}