1 Introduction

1.1 Goal

This httpserver.c program is to implement a simple single-thread Httpserver, receive the request and response to the client

1.2 Statement of Scope

The server will respond to simple GET and PUT commands to read and write files, and the command Head to provide information on existing files. With the following command, the server will respond to the validity of messages.

1.3 Running system

This program is running on the Linux system.

2 Data Design

2.1 Import function

This program needs to import each function by calling heads files
<sys/socket.h>, <sys/stat.h>, <stdio.h>, <netinet/in.h>, <netinet/ip.h>, <fcntl.h>, <err.h>, <errno.h>, <unistd.h>, <string.h>, <stdlib.h>, <stdbool.h>

2.2 Internal data

This program has a httpObject struct data type to store the component of messages from the user.

```
struct httpObject {
   /*
      Create some object 'struct' to keep track of all
      the components related to a HTTP message
      NOTE: There may be more member variables you would want to add
   */
   char method[5];         // PUT, HEAD, GET
   char filename[28];      // what is the file we are worried about
   char httpversion[9];    // HTTP/1.1
   ssize_t content_length; // example: 13
   int status_code;
   uint8_t buffer[BUFFER_SIZE];
};
```

2.3 Global data

BUFFER_SIZE is a global data to indicate the length of a buffer.

Buffer is a char array to store the message from the user.

3 Architectural and component-level design

3.1 System structure

In the main function, we first initialize the server_socket with address information from the command line, and listening to the incoming connection After checking the validity of server_socket, we create a client_socket connecting to the server within a while loop, in which the server keep waiting for the message from the client. In this loop, the server reads HTTP messages, processes requests, constructs httpresponse, and sends the response to the client.

3.11 read_http_response()

we send the client_sockd, and httpObject as arguments to this helper function: read_http_response(client_sockd, &message). In this function, we receive the http message and split it into different components and we store the different components into message struct.

ssize_t bytes = recv(client_sockd, buff, BUFFER_SIZE, 0)//suing recv to receive message and store it into buff

token=strtok(buff," ")//using function strtok to split message by the character

strcpy(message->method,token)//using strcpy to store the component of http message in the message struct

In the parsing of message, we have two situation, we may encounter Get and Head massage

```
GET /ABCDEFarqdeXYZxyzf012345-ab HTTP/1.1\r\n\r\n
HEAD /ABCDEFarqdeXYZxyzf012345-ab HTTP/1.1\r\n\r\n
```

We store the GET/HEAD in the message->method
We store /ABCDEFabcdef012345XYZxyz-mm in the message->filename
We store HTTP/1.1 in the message->httpversion

```
PUT /ABCDEFabcdef012345XYZxyz-mm HTTP/1.1\r\nContent-Length: 460\r\n\r\n
```

For PUT, we do the same thing above,moreover, if it is a Put message, we store the integer after the Content-Length into the message->content_length. After parsing the message, we return to main function and go to process_request function

3.12 process_request

After parsing the message, we start to process the request

First we check the validity of filename,

```
if(message->filename[i]>='a'&&message->filename[i]<='z'){
```

```
if(message->filename[i]>='A'&&message->filename[i]<='Z'){
```

```
if(message->filename[i]>='0'&&message->filename[i]<='9'){
```

```
if(message->filename[i]=='-'||message->filename[i]=='_')
```

After the message pass this checking, we begin to process the function,otherwise it report the 400 error to the client

```
sprintf(message->buffer,"HTTP/1.1 400 Bad Request\r\nContent-Length: 0\r\n\r\n");
send(client_sockd,message->buffer,strlen(message->buffer),0);
        close(client_sockd);
```

PUT Function

In this function, we send a file from client to server; we create a file given a filename; after the check the validity of the file we just created, we receive file data from the client, write the data in the file we just created.

Create a file

```
ssize_t file_descriptor =open(message->filename, O_WRONLY|O_CREAT|O_TRUNC,00700);
```

Check the file we created

```
                if(file_descriptor==-1){

f(errno==ENOENT){
                    sprintf(message->buffer,"HTTP/1.1 403 Forbideen\r\nContent-Length: 0\r\n\r\n");
                    send(client_sockd,message->buffer,strlen(message->buffer),0);
                   close(client_sockd);
                }

                   else{
                  sprintf(message->buffer,"HTTP/1.1 500 Internal Server Error\r\nContent-Length: 0\r\n\r\n");
                  send(client_sockd,message->buffer,strlen(message->buffer),0);
                   close(client_sockd);
```

Return 403 if we do not have access to the file, 500 otherwise.

We using a loop to write the data into the file

```
int counter=message->content_length;

while(counter>0){

        file_input=read(client_sockd, buff1,1);

       write(file_descriptor, buff1,file_input);

        counter=counter-1;

}
```

If the writing is success, we report 201 to the client and close file and client

```
if(file_input !=-1){

    dprintf(client_sockd,"HTTP/1.1 201 Created\r\nContent-Length: 0\r\n\r\n");
    //send(client_sockd,message->buffer,strlen(message->buffer),0);
     close(client_sockd);

}
```

HEAD FUNCTION

For this function, we just need to check the information of the file and return it to the client

Get the file

```
ssize_t file_descriptor =open(message->filename, O_RDONLY);
```

Check the status of file

```
if(file_descriptor==-1){
        if(errno==EACCES){

            sprintf(message->buffer,"HTTP/1.1 403 Forbideen\r\nContent-Length: %d\r\n\r\n",length);
            send(client_sockd,message->buffer,strlen(message->buffer),0);
            close(client_sockd);

        }
         else if(errno==ENOENT){
             sprintf(message->buffer,"HTTP/1.1 404 Not Found\r\nContent-Length: %d\r\n\r\n",length);
             send(client_sockd,message->buffer,strlen(message->buffer),0);
            close(client_sockd);
         }

            else{
          sprintf(message->buffer,"HTTP/1.1 500 Internal Server Error\r\nContent-Length: %d\r\n\r\n",length);
          send(client_sockd,message->buffer,strlen(message->buffer),0);
            close(client_sockd);


        }
```

Using the errono to check the error of file

Return 403 if we do not have access to the file, 404 if the file does not exist, 500 otherwise.

If the status file is good , we report 201 to the client and close file and client

```
if(file_descriptor !=-1){
   sprintf(message->buffer,"HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n",length);
       send(client_sockd,message->buffer,strlen(message->buffer),0);
       close(client_sockd);
 }
```

GET Function

For this function, we just need to check the information of the file and return the information and content of the file to the client.

Create a file

```
ssize_t file_descriptor =open(message->filename, O_WRONLY|O_CREAT|O_TRUNC,00700);
```

Check the file we created

```
                if(file_descriptor==-1){


f(errno==ENOENT){
                    sprintf(message->buffer,"HTTP/1.1 403 Forbideen\r\nContent-Length: 0\r\n\r\n");
                    send(client_sockd,message->buffer,strlen(message->buffer),0);
                  close(client_sockd);
                }

                  else{
                 sprintf(message->buffer,"HTTP/1.1 500 Internal Server Error\r\nContent-Length: 0\r\n\r\n");
                 send(client_sockd,message->buffer,strlen(message->buffer),0);
                  close(client_sockd);
```

Return 403 if we do not have access to the file, 500 otherwise.

We using a loop to write the data into the file

```
    }
        if(file_descriptor !=-1){
            sprintf(message->buffer,"HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n",length);
            send(client_sockd,message->buffer,strlen(message->buffer),0);
        }

        file_input=read(file_descriptor, buff1, sizeof(buff1)) ;
        while(file_input >0){
               write(client_sockd,buff1,file_input);
              file_input=read(file_descriptor, buff1,sizeof(buff1)) ;


        }
```

If the writing is success, we report 201 to the client and close file and client

```
if(file_input !=-1){

    dprintf(client_sockd,"HTTP/1.1 201 Created\r\nContent-Length: 0\r\n\r\n");
    //send(client_sockd,message->buffer,strlen(message->buffer),0);
     close(client_sockd);


}
```

4.0 user interface design

Users first call the server and give a port to the server, clients in the different directory send the message.

PUT instruction

```
curl -T client_input.txt http://localhost:8080/server_file
```

GET instruction

```
curl http://localhost:8080/server_file
```

HEAD instruction

```
curl -I http://localhost:8080/server_file
```

5.0 Testing Issues

All instructions perform well, noissue occur