Design Document: httpserver

Alan Li

1 Goals

The goal of this program is to implement an HTTP server. This server will respond to GET and PUT commands. When GET is requested, a file from the server is fetched. When PUT command is given, the client wants to send a file of a certain size on the server. The files will be indicated by 27 ASCII names. Essentially, our server will parse simple HTTP headers and provide a response. Status codes will also be implemented to dictate the status of the requests. (e.g 200 OK, 404 Error, etc..) It should also be able to handle incorrect requests and not crash.

2 Design

The several parts to handling the design of this project. The first part of the design is to handle opening a socket so that we can listen and accept requests on the server accordingly. After the socket is opened and requests can be parsed, we simply read or write respectively in accordance to commands. GET will fetch files while PUT will send a file from the client and they will have to be dealt with independently of each other. We must separate them and also make sure that we account for errors that may occur.

List of responses:

HTTP/1.1 400 BAD REQUEST

HTTP/1.1 403 FORBIDDEN

HTTP/1.1 404 NOT FOUND

HTTP/1.1 200 OK

HTTP/1.1 201 CREATED

HTTP/1.1 500 INTERNAL SERVER ERROR

2.1 Sockets

The program will first have to open a socket so that it can maintain a server client connection between to ends. The sockets will allow for the connection to be held and for messages to be sent from one to another. A hostname and port must be given to connect the two endpoints.

Input: Array of Arguments, argument count

```
If (argc>=3){
        //host name and port was given
        hostname= argv[1]
        port=argv[2]
}
if(argc==2){
        //no port was given,so we set to 80
        hostname=argv[1]
        port=80
}
if(argc==2){
        Throw error: no hostname
        exit;
}
Socket Creation (domain, type, protocol);
Bind() to bind socket to address and port number given;
Listen() to wait for client to make connection to or message server;
Accept() to get the connection request from client and deal with the messages;
```

2.2 Accepting client requests

We need to not close the server connection after one client request. We must leave it up so that they make more requests. We can implement using a while loop that continuously checks for accepts and at the end we will close the client socket

```
While(accept(main socket)){
        read(newSocket,buffer,buffer size)
        //Code for handling requests
        close(newSocket)
}
```

2.3 Handling Requests

The program will need a way to determine if the client has a GET or PUT request. This means that we first check the header to determine this, and respond accordingly depending on the message. We also will have to manage and see if the 27 ASCII names are valid (alphanumeric & "-" & "_"). If not, we will have to deal with that with messages. We will also have to ensure that the file can be read for the user to request it. If not, we must tell them that the file is forbidden.

Read from header save arguments in to buffers for use and reference
If (resource name is not 27 char or contains invalid char){
    Throw message to indicate problem/ BAD REQUEST
}
If (file cannot be read because it is forbidden (403)){
    Throw message to indicate problem/ FORBIDDEN
}
Otherwise if file exists:
Compare first buffer to "GET" and "PUT"
    If (GET)
        Deal with Get request from client;
        Give the file that was requested, throw errors if problem;
    If (PUT)
        Deal with Put request from client;
        Allow client to send file, throw errors if problem;
    Else it is an invalid request for our program;
If File doesn't exist{
    Throw 404 not found
}

2.4 Dealing with GET and PUT

Dealing with GET and PUT requests are somewhat different. GET will take a file from the server. PUT will let the client put a file on the server; they can also specify content length. PUT will take a file from the client and allow them to store it on the server in a file that they designate in their message as the resource name(27 ASCII). After dealing with the GET or PUT request, we must also indicate if the transaction was successful with a status message.

Input: files specified by requests
Create a buffer for use in requests
Create file with the correct resource name given
PUT:
Look for content length to see the size of the file the client wants to send.;

```
while(token!="Content-length"){
        if(token = "Content-length:"){
                length= next token
        }else{
                Go to next token
        }
}
Int totalRead
while(totalRead< length & read() == size of the buffer) {
        if(the amount of data to be read is <=32768){
                read(socket, buffer, length-totalRead)
                write(our file,buffer, length-totalRead)
                totalRead+=read()
        } else{
                read(socket, buffer, size of buffer)
                total+=read()
                write(our file,buffer, buffer size)
        }
}
```

Display message signifying e.g: 200 OK / 201 CREATED

GET:

```
//counting length
while(read!=0){
        len+=read();
        read(file,buffer,buffer size)
}
Throw a message indicating the status e.g: 200OK
Display the content length for the user

//printing the content
file=open(resourcename)
read(file , buffer)
While (read()!=0){
        write(socket,buffer)
        read(file, buffer, buffer size)
}
```