Design Document: HTTP Server

Aaron Nguyen
CruzID: anguy200

1 Goals

The goal of this program is to recreate the HTTP protocol in addition to have a functioning server in C or C++. The code should be able to mimic the functions of HTTP which is mainly being able to receive and send files usually code files and parse these to display information easily read by people.

2 Design

For this program there will be many parts to the code, and as the code is implemented more and more functions will be added or removed based on circumstances. The initial functions to be included into this doc will be a function that runs a server and the functions created in the last assignment dog which mimics the functionality of the Linux cat command.

2.1 Handling Arguments and Files

For this program HTTP, httpserver must be arg[0] before any file names. This checks if the function is actually being called and afterwards if it is true all subsequent file names are taken in. To determine the number of files give it would be the length of args minus one. After taking in the files, the files need to be parsed and be stored until all files have been parsed. To do this a malloc of up to 32 Kib can be used to store the data. After storing the data the second function is called to print the data.

```
#Input: file names
#Output: data found in each file
For length of args
     Declaration of socket variables
     Assign socket variables
     Get main socket ip and port
     Listen for connection requests
     while(accept true)
         Connect socket
         Receive Packets
         Call second function
```

2.2 Parsing Headers and Sending Responses

```
#Input: file names
#Output: data found in each file
For length of args
     if filename == -
          take stdin
          Print stdout()
     While recv is not 0
          Continue to receive packets
Call second function
```

2.3 Recombing and Printing Data to Socket or local Files

Due to all the data being stored in a heap. All that needs to be done is to pass the heap into a print function. Which will take each string and print it to the stdout

```
Printfile
     While (heap != empty)
     If not valid
          Return -1
     while string is not empty
          print
     return 1
     free heap
```

2.4 Checking Lines for Valid FileNames

The role of this function is to check whether or not the name of a file being requested or created is a proper file name

```
validity_check(string filename){
    char c;
    if(filename.length() != 27) return false;
    for( u_int i = 0; i < filename.length(); i++){
       c = filename.at(i);
       if((ascii letters, numbers or dash/underscore){
            continue;
```

```
        }else{
              return false;
        }
     }
     return true;
```

2.5 GET Method

Because there are two methods being used in this program this function is used to accomplish the GET task and check for errors while doing it

```
void get_parse(header, socket){
    if(file name is not valid){
       HTTP/1.1 400 Bad Request\r\n
    }
    Open file
    if(Insufficient permission is true){
       HTTP/1.1 403 Forbidden\r\n
       }
    if(if file doesn’t exist){

       HTTP/1.1 404 Not Found\r\n
   }else{
       Print function(file number, socket);
   }
```

2.6 PUT method

All this file does it makes sure the header is passing a valid file name for PUT to use and passes a variable to other functions to begin the PUT process

```
string put_parse(string header, int socket){
    if(if file name is valid){
       return temp;
    }
     HTTP/1.1 400 Bad Request\r\n
}
```

2.6 PUT method

All this file does it makes sure the header is passing a valid file name for PUT to use and passes a variable to other functions to begin the PUT process

```
int get_put_checker(string line){
    string temp = line.substr(0,3);
    if(temp == "GET"){
       return 1;
    }
    if(temp == "PUT"){
       return 2;
    }
    return 0;
}
```

2.7 Recombing and Printing Data to Socket or local Files

Due to all the data being stored in a heap. All that needs to be done is to pass the heap into a print function. Which will take each string and print it to the stdout

```
int catch_length(string line){
    if(line.substr(0,7).compare("Content") == 0){
       int first = (line.find(" ") + 1);
       string temp = line.substr(first);
       int size;
       size = stoi(temp);
       return size;
    }
    return -1;
}
```

2.9 Recombing and Printing Data to Socket or local Files

All this program does is goes through the initial lines sent in by the client and parses through it figure out what method the client is asking for. After discovering method type it calls up the appropriate method functions. This function is called repeatedly every time recv is called

```
void parse_recv(recv is recieving)
```

```
   Checks Method Type{
   if(Get){
      GET FUNCTION;
   }
   If(Put){
     PUT FUNCTION
   }
   Else{
     continue;
   }
}
```