Lab 1 Design Document

Shridhik John

Cruz ID: shjohn

CMPS 130, Fall 2019

1) Goal

The First goal is to set up my server. I will try working with/ creating a TCP server before I create an HTML server because I assume it will be easier to learn and understand. Then I'll be using Curl instead of the client to communicate with the server. As soon as this is achieved, I will work on using the system calls socket, bind, listen, accept, connect, send, recv, open, read, write, close. The server will respond to simple GET and PUT commands to read and write (respectively) "files" named by 27-character ASCII names. The server will persistently store files in a directory on the server, so it can be restarted or otherwise run on a directory that already has files.

2) Assumptions

I assume I can use the base code provided to us in lab and to create the server.c file. Since I cant use file calls, I assume that I can use string functions like sprintf() and sscanf() and printf() that aren't FILE * calls as mentioned on the assignment. I also assume I can use flags and O_CREAT, O_WRONLY to create open and read files. I assume I can use the reading and writing features from lab0 on this assignment as well for the GET and PUT command.

3) Design

My general approach for this lab was to first get the socket working. I

will go to lab and Geek for geeks for help on this part. Once I understand how to send and receive packets to the server, I'll focus on working on parsing through the header. I'll use strtok and split_response to access the buffer elements. If the first argument is Get, then I'll go to a get function, if the first argument is a Put argument I'll have a separate function for that. Then I'll check the second argument for the file name, and then see if I can read it/if it exists, and throw the corresponding error messages if applicable. Then I'll check to make sure the file is exactly 27 characters long with either "-", "_", or alpha/numeric values. If it's a get curl command, I'll read the filename, and send it to the curl command's folder. If it's a put command, I'll get the filename, content length, and recieve the information after I read and write it.

4) Pseudocode

```
void header(int handler, int status)
{// Check to see if file exists/is readable
        if exists send OK message
        else send Error 404
}


int main(int argc, char const *argv[])
{
//initialize buffer
// Connect socket code to connect client with server
```

insert Geek for Geeks code

```
//Go through every line in the header
while(split_response != NULL)
 {
printf("%s\n",split_response); // prints every line in header
split_response = strtok(NULL,"\r\n");
}
split_response = strtok(response, " "); // split every line in the header/response
  while(split_response != NULL)
    {
 //    printf("%s\n",split_response); // prints every line in header
      split_response = strtok(NULL,"\r\n");

    }
```

```
   split_response = strtok(response, " "); //  divde the buffer by space

// Split_response will store the information from the buffer on whether the call is a GET or PUT
command
//if response is a get
        // Get file name
         fname = strtok(NULL, " ");
         if (fname[0] == '/') fname++;

                //Check to see if file name is exactly 27 characters, consists of a hyphen,
                underscore, or alphanumeric value values
                // Check to see if file exists and is readable
                // If fails send 404 message
                // else continue

        send(new_socket, buf, strlen(buf), 0); // Send to client



//if response is a Put
        // Get file name
         fname = strtok(NULL, " ");
         if (fname[0] == '/') fname++;

                //Check to see if file name is exactly 27 characters, consists of a hyphen,
                underscore, or alphanumeric value values
                // Check to see if file exists and is readable
                // If fails send 404 message
                // else continue

// Open a file and read/Write with this line command from Dog.c
        fd = open(fname, O_CREAT | O_WRONLY | O_TRUNC);
        recv(new_socket, buf, 50, 0);
        write( fd, buf, 50);
        close(fd); // close file
```

5) Question

What happens in your implementation if, during a PUT with a `Content-Length`, the connection was closed, ending the communication early? This extra concern was not present in

your implementation of `dog`. Why not? Hint: this is an example of complexity being added by an extension of requirements (in this case, data transfer over a network).

ANSWER: I believe that if the connection is ended early, only what had time to be sent would be sent and everything else would be cut off. This was not a concern in our implementation of Dog because we had to manually enter in input ourselves on the command prompt rather than having it in a file or by sending it through a client/server.

•