Nathan Bender

CS 283 – Systems Programming

H3 – Network Programming

**Please note. All changes to the source code made for the questions is in bolded, red font.** I have also included the .c file for the entire tiny server.

1. 11.6
   1. Modify to echo requests

*I was unsure whether this question was asking to modify the requests and request headers to the terminal for the server, or to have the server echo the request and request headers back to the client. Therefore, I have done both.* ***PLEASE NOTE: Due to the echoing back to the client, the content no longer renders in the browser. I was told by the TA that this is fine since it was not part of the question.***

void doit(int fd)

{

int is\_static;

struct stat sbuf;

char buf[MAXLINE], method[MAXLINE], uri[MAXLINE], version[MAXLINE];

char filename[MAXLINE], cgiargs[MAXLINE];

rio\_t rio;

**ssize\_t linesRead;**

/\* Read request line and headers \*/

Rio\_readinitb(&rio, fd);

**linesRead =** Rio\_readlineb(&rio, buf, MAXLINE); //line:netp:doit:readrequest

**printf("Request Header:\n%s", buf);**

**rio\_writen(fd, buf, linesRead);**

sscanf(buf, "%s %s %s", method, uri, version);

if (strcasecmp(method, "GET")) {

clienterror(fd, method, "501", "Not Implemented",

"Tiny does not implement this method");

return;

}

read\_requesthdrs(&rio);

/\* Parse URI from GET request \*/

is\_static = parse\_uri(uri, filename, cgiargs);

if (stat(filename, &sbuf) < 0) {

clienterror(fd, filename, "404", "Not found",

"Tiny couldn't find this file");

return;

}

if (is\_static) { /\* Serve static content \*/

if (!(S\_ISREG(sbuf.st\_mode)) || !(S\_IRUSR & sbuf.st\_mode)) {

clienterror(fd, filename, "403", "Forbidden",

"Tiny couldn't read the file");

return;

}

serve\_static(fd, filename, sbuf.st\_size);

}

else { /\* Serve dynamic content \*/

if (!(S\_ISREG(sbuf.st\_mode)) || !(S\_IXUSR & sbuf.st\_mode)) {

clienterror(fd, filename, "403", "Forbidden",

"Tiny couldn't run the CGI program");

return;

}

serve\_dynamic(fd, filename, cgiargs);

}

}

void read\_requesthdrs(rio\_t \*rp)

{

char buf[MAXLINE];

**size\_t linesRead;**

**linesRead =** Rio\_readlineb(rp, buf, MAXLINE);

**printf("%s", buf);**

**rio\_writen(rp->rio\_fd, buf, linesRead);**

while(strcmp(buf, "\r\n")) { //line:netp:readhdrs:checkterm

**linesRead =** Rio\_readlineb(rp, buf, MAXLINE);

**printf("%s", buf);**

**rio\_writen(rp->rio\_fd, buf, linesRead);**

}

return;

}

* 1. Output of Tiny

**From server side:**

Request Header:

GET /home.html HTTP/1.1

Host: tux1.cs.drexel.edu:1736

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Cookie: \_\_utma=191177727.1035815306.1487090538.1487092743.1487096558.3; \_\_utmz=191177727.1487096558.3.3.utmcsr=learn.dcollege.net|utmccn=(referral)|utmcmd=referral|utmcct=/webapps/portal/execute/tabs/tabAction; \_ga=GA1.2.1035815306.1487090538

Connection: keep-alive

Upgrade-Insecure-Requests: 1

<!DOCTYPE html PUBLIC "-//w3c//dtd html 4.0 transitional//en">

<html>

<head>

<title>CS 283 -- Tiny Web Server</title>

</head>

<body>

<h1>CS 283 Systems Programming</h3>

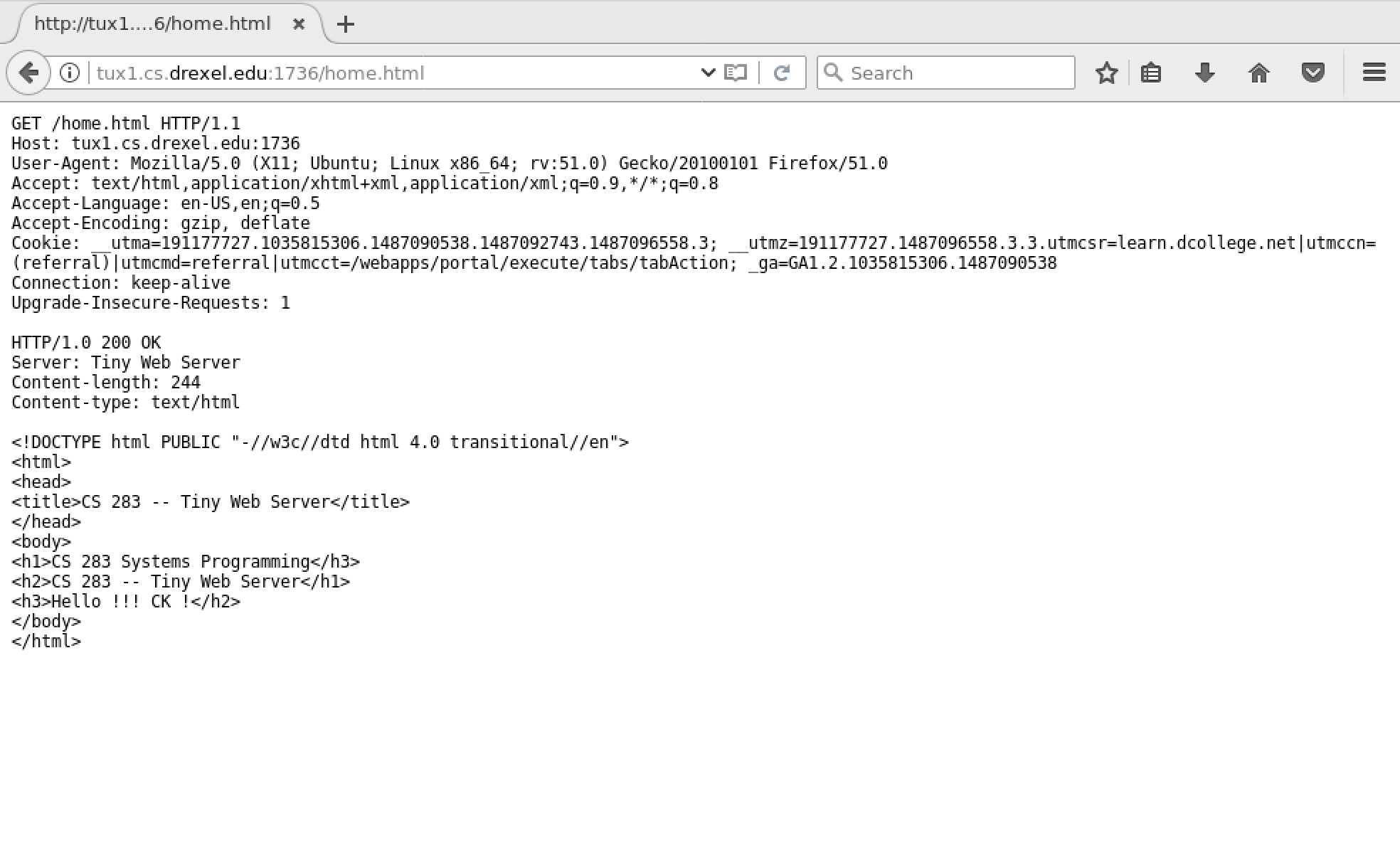
<h2>CS 283 -- Tiny Web Server</h1>

<h3>Hello !!! CK !</h2>

</body>

</html>

**From the client side:**

****

* 1. The browser uses HTTP version 1.1.

1. 11.7

In order to serve mpeg videos, all that was needed to be done was to add a file type to the get\_filetype function. This function copies the MIME type into the filetype variable. When accessing this URL in firefox, the mpeg video downloads, and I am prompted to open the video in my preferred video player.

void get\_filetype(char \*filename, char \*filetype)

{

if (strstr(filename, ".html"))

strcpy(filetype, "text/html");

else if (strstr(filename, ".gif"))

strcpy(filetype, "image/gif");

else if (strstr(filename, ".jpg"))

strcpy(filetype, "image/jpeg");

**else if (strstr(filename, ".mpg"))**

**strcpy(filetype, "video/mpeg");**

else

strcpy(filetype, "text/plain");

}

**Output from Tiny:**

Request Header:

GET /grb\_1.mpg HTTP/1.1

Host: tux1.cs.drexel.edu:1736

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Cookie: \_\_utma=191177727.1035815306.1487090538.1487092743.1487096558.3; \_\_utmz=191177727.1487096558.3.3.utmcsr=learn.dcollege.net|utmccn=(referral)|utmcmd=referral|utmcct=/webapps/portal/execute/tabs/tabAction; \_ga=GA1.2.1035815306.1487090538

Connection: keep-alive

Upgrade-Insecure-Requests: 1

1. 11.9

*I wasn’t sure how the book wanted me to use malloc, so I have included two solutions to this problem. The first one does not use malloc and works. I have also included a second version that uses malloc below the first (not sure if this is what was wanted though).*

In this solution, I have looped through the lines of the file, reading them one at a time. The lines are read into the buffer, and then written directly to the client file descriptor. This goes until the file has no more lines to read.

void serve\_static(int fd, char \*filename, int filesize)

{

int srcfd;

char \*srcp, filetype[MAXLINE], buf[MAXBUF];

/\* Send response headers to client \*/

get\_filetype(filename, filetype);

sprintf(buf, "HTTP/1.0 200 OK\r\n"); //line:netp:servestatic:beginserve

sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);

sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);

sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);

Rio\_writen(fd, buf, strlen(buf)); //line:netp:servestatic:endserve

**ssize\_t linesRead, linesWritten;**

**srcfd = Open(filename, O\_RDONLY, 0);**

**// While there are lines to be read, write them directly to the client**

**while((linesRead = rio\_readn(srcfd, buf, sizeof(buf))) != 0){**

**printf("%s", buf);**

**rio\_writen(fd, buf, linesRead);**

**}**

Close(srcfd);

}

In this solution, I used a temporary array that is the size of the file to store all of the bytes of the file. While there are still bytes in the file, I read the bytes from the file, store them in buf, and then copy the file contents into the temporary array. The pointer for the current position of the array is then updated. Once the entire file is in the array, we then write the file to the client file descriptor.

void serve\_static(int fd, char \*filename, int filesize)

{

int srcfd;

char \*srcp, filetype[MAXLINE], buf[MAXBUF];

/\* Send response headers to client \*/

get\_filetype(filename, filetype);

sprintf(buf, "HTTP/1.0 200 OK\r\n"); //line:netp:servestatic:beginserve

sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);

sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);

sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);

Rio\_writen(fd, buf, strlen(buf)); //line:netp:servestatic:endserve

**char \* content = malloc(filesize \* sizeof(char)); // create array**

**char \* content\_ptr = content; // point a pointer at beginning of array**

**ssize\_t linesRead, linesWritten;**

**srcfd = Open(filename, O\_RDONLY, 0);**

**// While there are lines still to be read, read them and store in buf**

**// Then, copy the contents of the buf into the array. Move the array**

**// pointer up.**

**while((linesRead = rio\_readn(srcfd, buf, sizeof(buf))) != 0){**

**memcpy(content\_ptr, buf, linesRead);**

**content\_ptr += linesRead;**

**}**

**rio\_writen(fd, content, filesize); //send the content to client**

Close(srcfd);

}