Design Document: Multi-Threaded HTTP Server with Logging

Julia Sales
Cruz ID: jesales

1) Goals

The goal for Assignment 2 is to modify Assignment 1, our HTTP server and add a multi-threaded HTTP server with logging. We take in the commands ex: **./httpserver -N 8 -l my_log.txt localhost 8888**. Where we have "N", which is the number of worker threads that the server uses. We also have "l", which logs the data of the requests and writes them into the file specified in the arguments. For this assignment, we must use condition variables, mutexes, and/or semaphores when implementing multi-threading.

2) Design

For this assignment, there are four parts to this assignment. The first part is that we need to set up the socket and server for our code, which we have already done in Assignment 1. The second part is modifying our code for Assignment 1 where we have to modify the PUT header, where we also take in content length. The third part is that we have to implement multi-threading. Implementing multi-threading will be broken up into more sub parts as we have to take in the argument that counts the number of worker servers and use synchronization mechanisms in our code. The last part is that we have to also implement the logging of the requests. This will also include multiple sub parts as well.

2.1 Setting Up The Socket

Creating a sockaddr_in

```
1. if argv[5] is not NULL then make it SERVER_NAME_STRING
2. if argv[6] is not NULL then make it PORT_NUMBER
3. if argv[5] is NULL print the error "Request is missing required `Host` header"
4. if argv[6] is NULL print the error "Request is missing required `Port` header"
5. struct hostent *hent = gethostbyname(SERVER_NAME_STRING /* eg
   "localhost" */);
2. struct sockaddr_in addr;
3. memcpy(&addr.sin_addr.s_addr, hent->h_addr, hent->h_length);
4. addr.sin_port = htons(PORT_NUMBER);
5. addr.sin_family = AF_INET;
```

Creating a Socket

```
1.  int sock = socket(AF_INET, SOCK_STREAM, 0);
2.  if no connection, when sock is 0
3.        |  Error: In socket, no connection
```

Socket Setup for Server

```
1.  int enable = 1;
2.  setsockopt(sock,  SOL_SOCKET,  SO_REUSEADOR,  &enable,  sizeof(enable));
3.  bind(sock,  (struct sockaddr *)&addr,  sizeof(addr));
4.        |  if not being able to bind - Error: cannot bind
5.  listen(sock,  0);
6.  int cl = accept(sock,  NULL,  NULL);
7.  if cl is < 0
8.        |  Error: Cannot accept
```

Parse Header

```
1.  create buffer with size 327868
2.  header1 = GET or  PUT <- the request
3.  header2 = the 27 ascii string <- the file
4.  header3 = HTTP/1,1
5.  header4 = “Content-length: %d”
6.  int valread = read(cl, buffer, size of buffer) <- buffer = header
7.  char line = strtok(strdup(buffer), newline);
8.  while line is not null
9.        |  if the first line
10.           |  use sscanf to put the request into header1, the target into header 2, and
    HTTP/1.1 into header 3
11.           |  if the first character of the target file is a slash, ignore the slash
12.                 |  use memmove here
13.       |  if line contains “Content-length: “
14.           |  put the content-length into header 4
15.       |  continue to next line
```

GET

```
1.  if the request command is “GET”
2.      | open the file <- file
3.      | if there is a “/“ in the target name
4.          | HTTP/1.1 403 Forbidden
5.          | Content-length: 0
6.      | if the file name is not 27 ascii characters long
7.          | HTTP/1.1 400 Bad Request
8.          | Content-length: 0
9.      | if file is not found <- if the open file returns -1
10.         | HTTP/1.1 404 Not Found
11.         | Content-length: 0
12.     | if file is found <- if open file is not equal to -1
13.         | read file = read(file, file buffer, buffer size);
14.         | get Content-length (2.3b)
15.         | HTTP/1.1 200 OK
16.         | print out the content length
17.         | while file size is >= size of buffer <- handling large files
18.             | write out contents from first read
19.             | read file again starting at file
20.         | write remaining bits
21.      | else
22.         | HTTP/1.1 500 Internal Server Error
23.         | Content-length: 0
```

Get Content Length

```
1.  struct stat st;
2.  stat(target, st);
3.  content-length = st.st_size;
```

Close the connection

Closes at the end of the while loop, while(1)

```
At the end of everything
close(cl);
```

2.2 Modify the PUT Header

PUT

```
if content length is not provided
    | HTTP/1.1 400 Bad Request
    | Content-length: 0
if file contains “/“
    | HTTP/1.1 403 Forbidden
    | Content-length: 0
else if target name is not 27 ascii characters long
    | HTTP/1.1 400 Bad Request
    | Content-length: 0
else
    | fd = open and create the file as header2
    | if file cannot be created
        | HTTP/1.1 500 Internal Sever Error
        | close fd
    | else
        | HTTP/1.1 100 Continue
        | if the content length is smaller than the buffer
            | putread = read(from cl);
            | write(to fd);
        | else
            | putread = read(from cl);
            | write(to fd);
            | bytes_read = content length - buffer size
            | while bytes_read is greater than or equal to buffer size
                | putread = read from cl again
                | write to fd again
                | bytes_read = bytes_read - buffer size
            | putread = read from cl last time
            | write to fd with remaining bytes
    | HTTP/1.1 201 Created
    | Content-length: 0
    | close fd
```