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. sscanf(buffer, “%s %s %s”, header1, header2, header3); <- how we parse the
   header
8. char line = strtok(strdup(buffer), newline);
9. while line is not null
10.      |  if the first line
11.           |  sscanf(line, “%s %s %s”, header1, header2, header3);
12.      |  if line contains “Content-length: “
13.           |  sscanf(line, “%s %d”, header4, content-length);
14.      |  line = strtok(NULL, new line);
```

GET

```
1.  if the request command is “GET”
2.      | open the file <- file
3.      | if there is a “/“ in the target name
4.          | write(cl, HTTP/1.1 403 Forbidden);
5.          | Content-length: 0
6.      | if the file name is not 27 ascii characters long
7.          | write(cl, HTTP/1.1 400 Bad Request);
8.          | Content-length: 0
9.      | if file is not found <- if the open file returns -1
10.         | write(cl, 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.         | write(cl, HTTP/1.1 200 OK);
16.         | sprintf(new buffer, content-length: %d, content-length);
17.         | write(cl, content-length);
18.         | while file size is >= size of buffer <- handling large files
19.             | write(cl, file contents);
20.             | read file again starting at file
21.         | write(cl, file contents at file buffer)
22.     | else
23.         | write(cl, HTTP/1.1 500 Internal Server Error);
24.         | 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

```
1.  if file contains "/"
2.      | write(cl, HTTP/1.1 403 Forbidden);
3.      | Content-length: 0
4.  else if target name is not 27 ascii characters long
5.      | write(cl, HTTP/1.1 400 Bad Request);
6.      | Content-length: 0
7.  else
8.      | fd = open(header2, O_RDWR |O_CREAT|O_TRUNC, 0664);
9.      | if file cannot be created
10.         | write(cl, HTTP/1.1 500 Internal Sever Error);
11.         | close fd
12.     | else
13.         | write(cl, HTTP/1.1 100 Continue);
14.         | if the content length is smaller than the buffer
15.             | putread = read(from cl);
16.             | write(to fd);
17.         | else
18.             | putread = read(from cl);
19.             | write(to fd);
20.             | bytes_read = content length - buffer size
21.             | while bytes_read is greater than or equal to buffer size
22.                 | putread = read(from cl again);
23.                 | write(to fd again);
24.                 | bytes_read = bytes_read - buffer size
25.             | putread = read(from cl last time);
26.             | write(to fd with remaining bytes);
27.     | write(cl, HTTP/1.1 201 Created);
28.     | close fd
```