浙江大学

本科实验报告

课程名称：　计算机网络基础

实验名称：　实现一个轻量级的 WEB 服务器

姓　　名：

学　　院：　计算机学院

系：　计算机系

专　　业：　计算机科学与技术

学　　号：

指导教师：　张泉方

2022 年　11 月　20 日

浙江大学实验报告

实验名称：实现一个轻量级的 WEB 服务器　实验类型：编程实验

同组学生：无　实验地点：计算机网络实验室

一、 实验目的

深入掌握 HTTP 协议规范，学习如何编写标准的互联网应用服务器。

二、 实验内容

- 服务程序能够正确解析 HTTP 协议，并传回所需的网页文件和图片文件
- 使用标准的浏览器，如 IE、Chrome 或者 Safari，输入服务程序的 URL 后，能够正常显示服务器上的网页文件和图片
- 服务端程序界面不做要求，使用命令行或最简单的窗体即可
- 功能要求如下：
  1. 服务程序运行后监听在 80 端口或者指定端口
  2. 接受浏览器的 TCP 连接（支持多个浏览器同时连接）
  3. 读取浏览器发送的数据，解析 HTTP 请求头部，找到感兴趣的部分
  4. 根据 HTTP 头部请求的文件路径，打开并读取服务器磁盘上的文件，以 HTTP 响应格式传回浏览器。要求按照文本、图片文件传送不同的 Content-Type，以便让浏览器能够正常显示。
  5. 分别使用单个纯文本、只包含文字的 HTML 文件、包含文字和图片的 HTML 文件进行测试，浏览器均能正常显示。
- 本实验可以在前一个 Socket 编程实验的基础上继续，也可以使用第三方封装好的 TCP 类进行网络数据的收发
- 本实验要求不使用任何封装 HTTP 接口的类库或组件，也不使用任何服务端脚本程序如 JSP、ASPX、PHP 等

三、 主要仪器设备

联网的 PC 机、Wireshark 软件、Visual Studio、gcc 或 Java 集成开发环境。

四、 操作方法与实验步骤

- 阅读 HTTP 协议相关标准文档，详细了解 HTTP 协议标准的细节，有必要的话使用 Wireshark 抓包，研究浏览器和 WEB 服务器之间的交互过程
- 创建一个文档目录，与服务器程序运行路径分开
- 准备一个纯文本文件，命名为 test.txt，存放在 txt 子目录下面
- 准备好一个图片文件，命名为 logo.jpg，放在 img 子目录下面
- 写一个 HTML 文件，命名为 test.html，放在 html 子目录下面，主要内容为：

```
<html>
    <head><title>Test</title></head>
    <body>
        <h1>This is a test</h1>
        <img src="img/logo.jpg">
        <form action="dopost" method="POST">
            Login:<input name="login">
            Pass:<input name="pass">
            <input type="submit" value="login">
        </form>
    </body>
</html>
```

- 将 test.html 复制为 noimg.html，并删除其中包含 img 的这一行。
- 服务端编写步骤（**需要采用多线程模式**）

a) 运行初始化，打开 Socket，监听在指定端口（**请使用学号的后 4 位作为服务器的监听端口**）

b) 主线程是一个循环，主要做的工作是等待客户端连接，如果有客户端连接成功，为该客户端创建处理子线程。该子线程的主要处理步骤是：

1. 不断读取客户端发送过来的字节，并检查其中是否连续出现了 2 个回车换行符，如果未出现，继续接收；如果出现，按照 HTTP 格式解析第 1 行，分离出方法、文件和路径名，其他头部字段根据需要读取。

✧ **如果解析出来的方法是 GET**

2. 根据解析出来的文件和路径名，读取响应的磁盘文件（该路径和服务器程序可能不在同一个目录下，需要转换成绝对路径）。如果文件不存在，第 3 步的响应消息的状态设置为 404，并且跳过第 5 步。
3. 准备好一个足够大的缓冲区，按照 HTTP 响应消息的格式先填入第 1 行（状态码=200），加上回车换行符。然后模仿 Wireshark 抓取的 HTTP 消息，填入必要的几行头部（需要哪些头部，请试验），其中不能缺少的 2 个头部是 Content-Type 和 Content-Length。Content-Type 的值要和文件类型相匹配（请通过抓包确定应该填什么），Content-Length 的值填写文件的字节大小。
4. 在头部行填完后，再填入 2 个回车换行
5. 将文件内容按顺序填入到缓冲区后面部分。

✧ **如果解析出来的方法是 POST**

6. 检查解析出来的文件和路径名，如果不是 dopost，则设置响应消息的状态为 404，然后跳到第 9 步。如果是 dopost，则设置响应消息的状态为 200，并继续下一步。
7. 读取 2 个回车换行后面的体部内容（长度根据头部的 Content-Length 字段的指示），并提取出登录名（login）和密码（pass）的值。如果登录名是你的学号，密码是学号的后 4 位，则将响应消息设置为登录成功，否则将响应消息设置为登录失败。
8. 将响应消息封装成 html 格式，如

<html><body>响应消息内容</body></html>

9. 准备好一个足够大的缓冲区，按照 HTTP 响应消息的格式先填入第 1 行（根据前面的情况设置好状态码），加上回车换行符。然后填入必要的几行头部，其中不能缺少的 2 个头部是 Content-Type 和 Content-Length。Content-Type 的值设置为 text/html，如果状态码=200，则 Content-Length 的值填写响应消息的字节大小，并将响应消息填入缓冲区的后面部分，否则填写为 0。
10. 最后一次性将缓冲区内的字节发送给客户端。
11. 发送完毕后，关闭 socket，退出子线程。

c) 主线程还负责检测退出指令（如用户按退出键或者收到退出信号），检测到后即通知并等待各子线程退出。最后关闭 Socket，主程序退出。

- 编程结束后，将服务器部署在一台机器上（本机也可以）。在服务器上分别放置纯文本文件（.txt）、只包含文字的测试 HTML 文件（将测试 HTML 文件中的包含 img 那一行去掉）、包含文字和图片的测试 HTML 文件（以及图片文件）各一个。
- 确定好各个文件的 URL 地址，然后使用浏览器访问这些 URL 地址，如 http://x.x.x.x:port/dir/a.html，其中 port 是服务器的监听端口，dir 是提供给外部访问的路径，请设置为与文件实际存放路径不同，通过服务器内部映射转换。
- 检查浏览器是否正常显示页面，如果有问题，查找原因，并修改，直至满足要求
- 使用多个浏览器同时访问这些 URL 地址，检查并发性

五、 实验数据记录和处理

请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **源代码：需要说明编译环境和编译方法，如果不能编译成功，将影响评分**
- **可执行文件：可运行的.exe 文件或 Linux 可执行文件**

- 服务器的主线程循环关键代码截图（解释总体处理逻辑，省略细节部分）

```
while (!serverExit) {
    // accept new client
    printPrompt("Waiting for new client...");
    cout << endl;
    struct sockaddr_in clnt_addr;
    socklen_t clnt_addr_len = sizeof(clnt_addr);
    bzero(&clnt_addr, sizeof(clnt_addr));
    int clnt_sockfd;

    setjmp(jmpbuf);
    if(serverExit) {
        break;
    }
    clnt_sockfd = accept(sevSockfd, (sockaddr*)&clnt_addr, &clnt_addr_len);
    errif(clnt_sockfd == -1, "socket accept error");

    char ipAddr[IPV4_LEN];
    uint16_t port;
    bzero(ipAddr, IPV4_LEN);
    strcpy(ipAddr, inet_ntoa(clnt_addr.sin_addr));
    port = ntohs(clnt_addr.sin_port);
    cout << PROMPT << "New client: [IPAddress]" << ipAddr << " [Port]" << port << endl;

    // Create a child thread to handle this client
    pool.enqueue(handleClient, clnt_sockfd);
}
```

主线程等待新客户的连接，accept 接受新客户后为其分配新的 socket fd。然后向线程池的任务队列中添加新任务，让子线程去处理客户请求。

- 服务器的客户端处理子线程关键代码截图（解释总体处理逻辑，省略细节部分）

```
// get request
cout << GREEN << "Request:" << endl;
readLine(clnt_sockfd, buf, sizeof(buf));
cout << buf;

// parse method, url (GET /test.html HTTP/1.1)
sscanf(buf, "%s %s", method, url);
```

首先读取请求头的第一行，并分离出方法和路径，然后针对不同方法分别处理。

```
// method = GET
if (strcasecmp(method, "GET") == 0) {
    discardHeader(clnt_sockfd);
    cout << WHITE;
    // convert url to actual file path && get file type
    url2Path(url, path, filetype);
    // cout << endl << url << endl;
    // cout << path << endl;

    if (stat(path, &st) == -1) {
        response(clnt_sockfd, "404", "NOT FOUND", "The requested resource not found");
    } else if (!(S_IRUSR & st.st_mode)) {
        response(clnt_sockfd, "403", "FORBIDDEN", "The requested resource inaccessible");
    } else {
        if ((st.st_mode & S_IFMT) == S_IFDIR) {
            strcat(path, "/index.html");
            strcpy(filetype, "text/html");
        }

        response(clnt_sockfd, path, st.st_size, filetype);
    }
}
```

若是 GET 方法，则将浏览器中的 url 映射成服务器本地的实际路径，如果该文件存在且可读，则组装响应头以及响应体(即文件内容)并发送给客户。如果找不到该文件，则响应 404。

```
// method = POST
else if (strcasecmp(method, "POST") == 0) {
    // the url is /dopost
    if (strcmp(url, "/dopost") == 0) {

        char line[BUFFERSIZE];
        int len;
        // discard the header and get Content-Length
        discardHeader(clnt_sockfd, line, "Content-Length");
        sscanf(line, "Content-Length: %d", &len);

        // read the body
        readLine(clnt_sockfd,buf, len);
        cout << buf << endl << endl << WHITE;
        // check username and password
        if(strcmp(buf,"login=3200103936&pass=3936") == 0) {
            response(clnt_sockfd, "200", "OK", "Login successfully");
        } else {
            response(clnt_sockfd, "200", "OK", "Login failed");
        }
    }
    else {
        response(clnt_sockfd, "404", "NOT FOUND", "Not dopost");
    }
}
else {
    response(clnt_sockfd, "501", "METHOD NOT IMPLEMENTED", "Request method not supported");
}
```

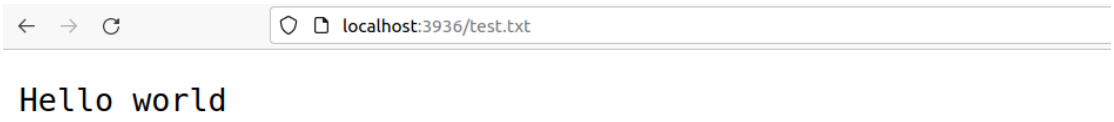
若是 POST 方法，如果 url 为 /dopost，则根据账号密码是否匹配响应登录状态，否则响应 404。对于其它方法，响应 501 未实现。

- 服务器运行后，用 netstat –an 显示服务器的监听端口

```
(base) yyb@yyb-laptop:~/zju/cn22fall$ sudo netstat -anp | grep 3936
[sudo] password for yyb:
tcp        0      0 127.0.0.1:3936          0.0.0.0:*               LISTEN      34958/./server
```

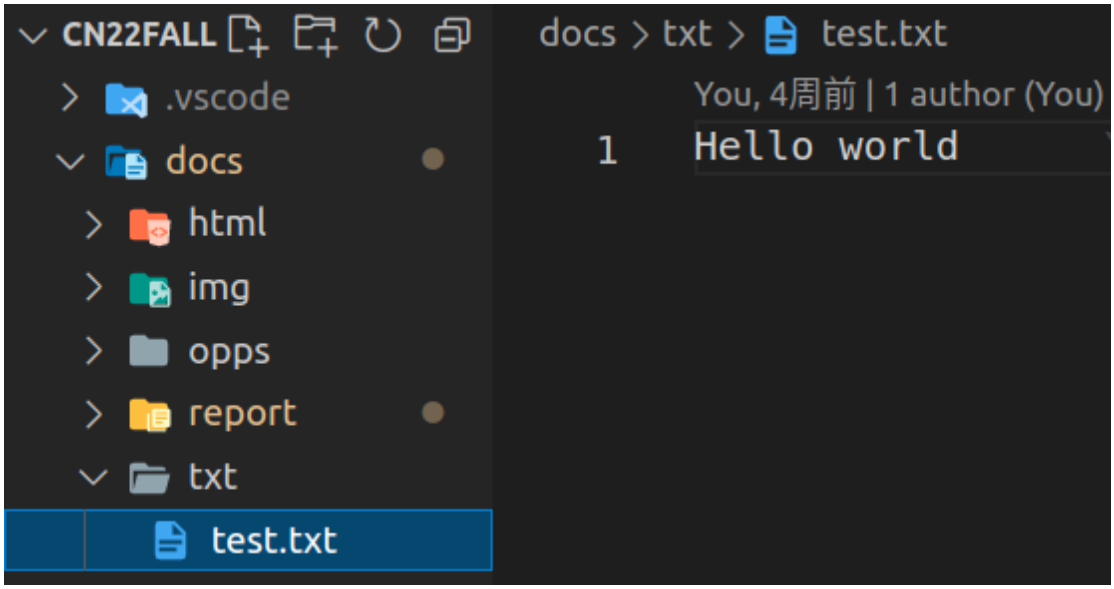
如图，监听端口为 3936，是学号后四位。

- 浏览器访问纯文本文件（.txt）时，浏览器的 URL 地址和显示内容截图。



Hello world

服务器上文件实际存放的路径：



服务器的相关代码片段：

```
// method = GET
if (strcasecmp(method, "GET") == 0) {
    discardHeader(clnt_sockfd);
    cout << WHITE;
    // convert url to actual file path && get file type
    url2Path(url, path, filetype);
    // cout << endl << url << endl;
    // cout << path << endl;

    if (stat(path, &st) == -1) {
        response(clnt_sockfd, "404", "NOT FOUND", "The requested resource not found");
    } else if (!(S_IRUSR & st.st_mode)) {
        response(clnt_sockfd, "403", "FORBIDDEN", "The requested resource inaccessible");
    } else {
        if ((st.st_mode & S_IFMT) == S_IFDIR) {
            strcat(path, "/index.html");
            strcpy(filetype, "text/html");
        }

        response(clnt_sockfd, path, st.st_size, filetype);
    }
}
```

```
void url2Path(char* url, char* path, char* filetype) {

    if(url[strlen(url) - 1] == '/') {
        sprintf(path, "docs/html/test.html");
        strcpy(filetype, "text/html");
        return;
    }

    if (strstr(url, ".html")) {
        sprintf(path, "docs/html%s", url);
        strcpy(filetype, "text/html");
    }
    else if (strstr(url, ".png")) {
        sprintf(path, "docs/img%s", url);
        strcpy(filetype, "image/png");
    }
    else if (strstr(url, ".jpg")) {
        sprintf(path, "docs/img%s", url);
        strcpy(filetype, "image/jpeg");
    }
    else if (strstr(url, ".txt")) {
        sprintf(path, "docs/txt%s", url);
        strcpy(filetype, "text/plain");
    } else {
        sprintf(path, "docs%s", url);
    }
}
```

根据 url 中的文件后缀名信息设置文件类型，并将 url 映射成本地实际路径。

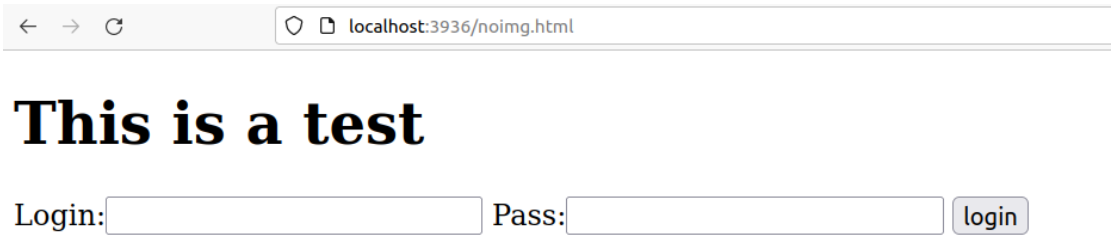
```cpp
// Response with file
void response(int sockfd, const char *filepath, int filesize, char* filetype) {

    char buf[BUFFERSIZE];
    cout << YELLOW << "Response:" << endl;

    // HEADER
    strcpy(buf, "HTTP/1.1 200 OK\r\n");
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    strcpy(buf, "Server: yey/1.0\r\n");
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    sprintf(buf, "Content-Type: %s\r\n", filetype);
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    sprintf(buf, "Content-Length: %d\r\n\r\n", filesize);
    send(sockfd, buf, strlen(buf), 0);
    cout << buf << WHITE;

    // BODY
    int resfd = open(filepath, O_RDONLY, 0);
    char* res = (char*)mmap(0, filesize, PROT_READ, MAP_PRIVATE, resfd, 0);
    close(resfd);
    send(sockfd, res, filesize, 0);
    munmap(res, filesize);
}
```

组装响应头和响应体，并发送。

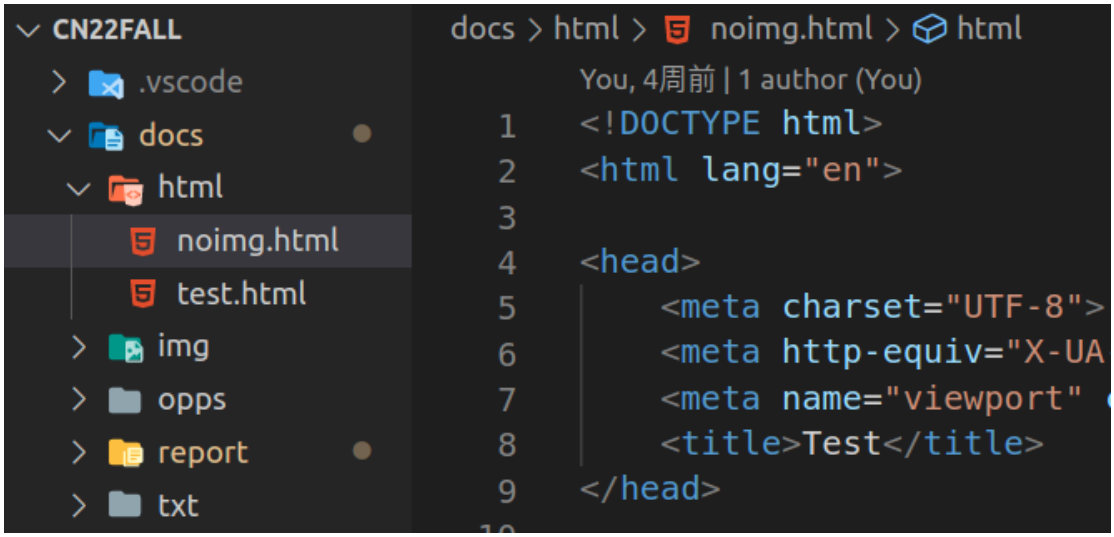
Wireshark 抓取的数据包截图（通过跟踪 TCP 流，只截取 HTTP 协议部分）：

```
GET /test.txt HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

HTTP/1.1 200 OK
Server: yey/1.0
Content-Type: text/plain
Content-Length: 11

Hello world
```

- 浏览器访问只包含文本的 HTML 文件时，浏览器的 URL 地址和显示内容截图。



服务器文件实际存放的路径：

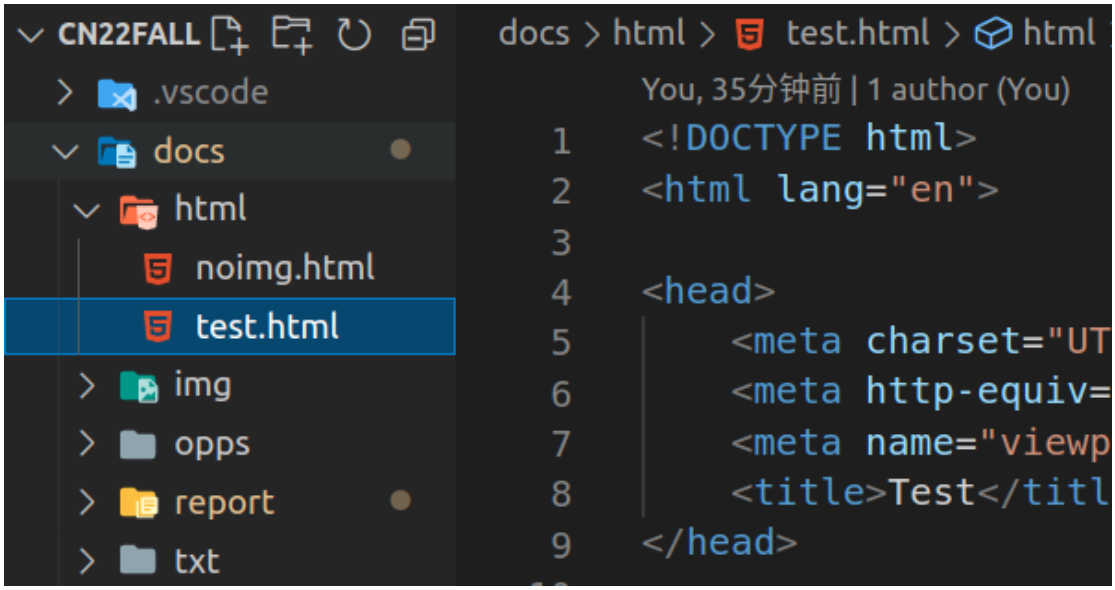


Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML 内容）：

```
GET /noimg.html HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

HTTP/1.1 200 OK
Server: yey/1.0
Content-Type: text/html
Content-Length: 450

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
</head>

<body>
    <h1>This is a test</h1>
    <form action="dopost" method="POST">
        Login:<input name="login">
        Pass:<input name="pass">
        <input type="submit" value="login">
    </form>
</body>

</html>
```

- 浏览器访问包含文本、图片的 HTML 文件时，浏览器的 URL 地址和显示内容截图。

服务器上文件实际存放的路径：

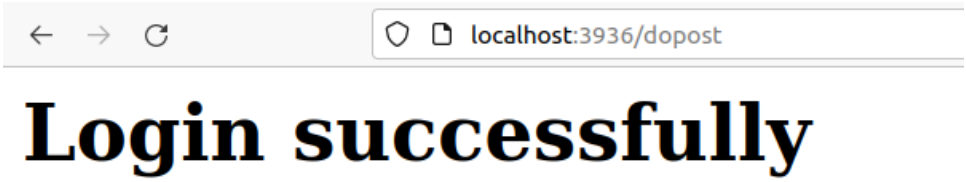


Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML、图片文件的部分内容）：

```
GET /test.html HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

HTTP/1.1 200 OK
Server: yey/1.0
Content-Type: text/html
Content-Length: 475

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test</title>
</head>

<body>
    <h1>This is a test</h1>
    <img src="logo.png">
    <form action="dopost" method="POST">
        Login:<input name="login">
        Pass:<input name="pass">
        <input type="submit" value="login">
    </form>
</body>

</html>
```

```
GET /logo.png HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:3936/test.html
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin

HTTP/1.1 200 OK
Server: yey/1.0
Content-Type: image/png
Content-Length: 149890

.PNG
.
...
IHDR.....................IDATx...Y.$Yv.v....{,.gV..{703..$..f..h&.Qz.....D..Afx.M2.......P....`.^......r..X...sdwq.....g...
8.......G.........3X....   ....$..DYx=........$......+Q.;.....Wn...`e+[..^...>....Y..[]..V..f+.Z..7y...m!.|:z.Wa...=..]M[...^..
\.....9S...l....1.".Ix..-Y..K..y..0Vv.l.X..6_.W1..6e...j..y".D..b..%.N.<..q.*..a...^T...k.....VvEl.X.
.....#....~.0xC...J...buO.@.5..>.....i".
...an..b.P8}$..yi.qe..V..M3.F...a..S.............._;.Ipx./..?..A..CD>~C!n"AP.W..!...[.0..0.f%..&.{R..y............#.xe.p...*.
.....$iP ...........&................zo..BAY.K...-............!.A..F...,....r....f.R.f..:v\;f......I..x.....gy....&"BD.X..Rd..
5j."0F...2.$:........Ni.IZ..b.......U.....`}.........!............|L3[. .!wX.VZ....35.....p..w....v...B...0.`........
$.b,Oa.a's..f..#8..).9i4.XW.....k........>>...-....|tV..rV..&.(...C...........  ........*....vl....Y.o.....s...Ba......2C.9230...
\.zO.
.h..Z..i.*.....q ..         ZP;PMJ..{.d.....WL.....@1.c.........\..
g...K.Y..H...`H.:5..{.B..~|z.....[.^...]...x..d.R......4c.......~....=@q...
.......|.f...G......N..u...........G......~.7(.......].4....pC....RQ.)..\..
P..C.lk.......htVU.....Z.i5..k..j:.N..... /.L.s.....efT1..~......F.....JI-\!K@o...br.:u..`..m.X_+.!.1.u.....v.R..df.....
G.RR..1
```

- 浏览器输入正确的登录名或密码，点击登录按钮（login）后的显示截图。



服务器相关处理代码片段：

```
// method = POST
else if (strcasecmp(method, "POST") == 0) {
    // the url is /dopost
    if (strcmp(url, "/dopost") == 0) {

        char line[BUFFERSIZE];
        int len;
        // discard the header and get Content-Length
        discardHeader(clnt_sockfd, line, "Content-Length");
        sscanf(line, "Content-Length: %d", &len);

        // read the body
        readLine(clnt_sockfd,buf, len);
        cout << buf << endl << endl << WHITE;
        // check username and password
        if(strcmp(buf,"login=3200103936&pass=3936") == 0) {
            response(clnt_sockfd, "200", "OK", "Login successfully");
        } else {
            response(clnt_sockfd, "200", "OK", "Login failed");
        }
    }
    else {
        response(clnt_sockfd, "404", "NOT FOUND", "Not dopost");
    }
}
```

```
// Response without file
void response(int sockfd, const char* sCode, const char* rPhrase, const char* content) {

    char buf[BUFFERSIZE];
    char body[BUFFERSIZE];

    cout << YELLOW << "Response:" << endl;
    sprintf(body, "<HTML><TITLE>%s</TITLE><BODY><H1>%s</H1></BODY></HTML>", rPhrase, content);

    sprintf(buf, "HTTP/1.1 %s %s\r\n", sCode, rPhrase);
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    sprintf(buf, "Server: yey/1.0\r\n");
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    sprintf(buf, "Content-Type: text/html\r\n");
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    sprintf(buf, "Content-Length: %d\r\n\r\n", (int)strlen(body)); // length is important
    send(sockfd, buf, strlen(buf), 0);
    cout << buf;
    strcpy(buf, body);
    send(sockfd, buf, strlen(buf), 0);
    cout << buf << endl << endl << WHITE;
}
```

Wireshark 抓取的数据包截图（HTTP 协议部分）

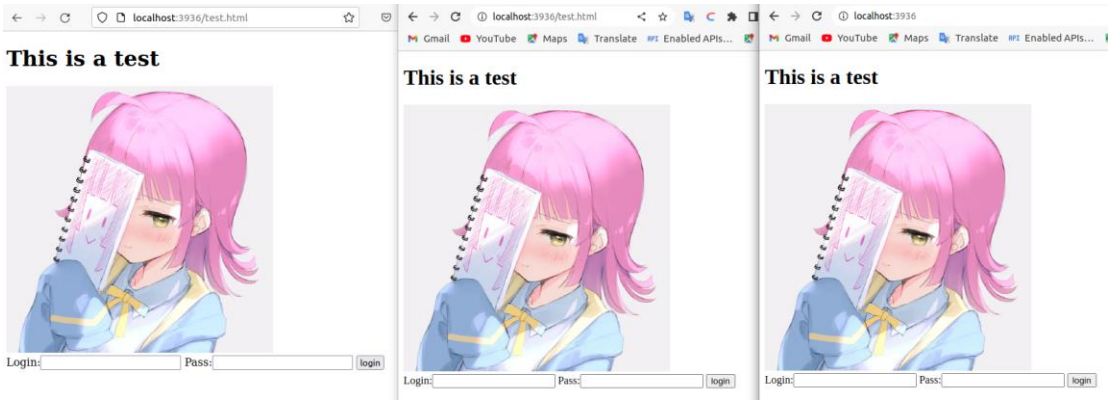
```
POST /dopost HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Origin: http://localhost:3936
Connection: keep-alive
Referer: http://localhost:3936/test.html
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

login=3200103936&pass=3936HTTP/1.1 200 OK
Server: yey/1.0
Content-Type: text/html
Content-Length: 70

<HTML><TITLE>OK</TITLE><BODY><H1>Login successfully</H1></BODY></HTML>
```

- 浏览器输入错误的登录名或密码，点击登录按钮（login）后的显示截图。

- Wireshark 抓取的数据包截图（HTTP 协议部分）

```
POST /dopost HTTP/1.1
Host: localhost:3936
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: http://localhost:3936
Connection: keep-alive
Referer: http://localhost:3936/test.html
Cookie: G_ENABLED_IDPS=google; io=BWohRTQSEW88TNIAAAAG; G_AUTHUSER_H=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

login=3200103936&pass=3936dHTTP/1.1 200 OK
Server: yey/1.0
Content-Type: text/html
Content-Length: 64

<HTML><TITLE>OK</TITLE><BODY><H1>Login failed</H1></BODY></HTML>
```

- 多个浏览器同时访问包含图片的 HTML 文件时，浏览器的显示内容截图（将浏览器窗口缩小并列）



- 多个浏览器同时访问包含图片的 HTML 文件时，使用 netstat –an 显示服务器的 TCP 连接（截取与服务器监听端口相关的）

```
tcp        0      0 127.0.0.1:3936          0.0.0.0:*               LISTEN      34958/./server
tcp        0      0 127.0.0.1:3936          127.0.0.1:34106         TIME_WAIT   -
tcp        0      0 127.0.0.1:3936          127.0.0.1:34100         TIME_WAIT   -
tcp        0      0 127.0.0.1:3936          127.0.0.1:34102         TIME_WAIT   -
tcp        0      0 127.0.0.1:3936          127.0.0.1:34096         TIME_WAIT   -
tcp        0      0 127.0.0.1:3936          127.0.0.1:34104         TIME_WAIT   -
```

六、 实验结果与分析

- HTTP 协议是怎样对头部和体部进行分隔的？

  通过空行进行分割

- 浏览器是根据文件的扩展名还是根据头部的哪个字段判断文件类型的？

  根据头部的 Content-Type 字段来判断文件类型，例如 img/png 对应 png 格式的图片文件

- HTTP 协议的头部是不是一定是文本格式？体部呢？

  头部一定是文本格式，体部可以是文本也可以是图片等字节流(二进制)格式。

- POST 方法传递的数据是放在头部还是体部？两个字段是用什么符号连

接起来的？

数据放在体部，通过&符号连接

七、 讨论、心得

本次实验的服务器建立在实验二的基础上，所以和 socket 相关的代码已经完成，主要就是处理 HTTP 报文。通过在 wireshark 上抓包就可以看到 HTTP 报文的格式，包括请求格式和响应格式，然后主要就是进行字符处理，在接收到的请求中分离出需要的内容，做出相应处理。响应也主要是根据格式设置好各个字段然后发送，在发送二进制文件时需要注意不能使用处理文本文件的函数，必须以字节流形式读取和发送数据。助教之前提到了线程池，所以想要将原来不太安全的 vector 换成线程池，但是对多线程的编程还不够熟悉，所以参考了一些项目，学习了线程池的管理，收获很大。在退出服务器时遇到了问题，为了安全关闭各个线程，需要在线程池的析构中 join 各个子线程，但是发现有些线程始终处于阻塞状态，无法正常退出，后来发现是谷歌浏览器会额外发起多个 socket 连接，导致部分线程被占用且阻塞，暂时还不知道怎么处理，但是换成火狐浏览器则可以安全关闭，因为火狐浏览器针对一个 HTTP 请求只会建立一个连接，响应请求后对应线程就能正常关闭。