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）

服务器的主线程循环接受来自客户端的连接，一旦接收到连接请求，创建子线程用于接受来自该连接的消息，而主线程继续等待新的连接。

```
1.  while (1) {
2.          AcceptSocket = SOCKET_ERROR;
3.          while (AcceptSocket == SOCKET_ERROR) {
4.              //keep accepting connection request from port 4112
5.              AcceptSocket = accept(MainSocket, NULL, NULL);
6.          }
7.          //when receive a new connection
8.          pThreadData PThreadData = (pThreadData)malloc(sizeof(ThreadData));
9.          PThreadData->AcceptSocket = AcceptSocket;
10.         memcpy(&PThreadData->server_conf, &server_conf, sizeof(s_conf));
11.         HANDLE hThread;
```

```
        DWORD  threadId;
        // create thread to process the request received
        hThread = CreateThread(NULL, 0, ThreadProcess, PThreadData, 0, &threadId);
    }
```

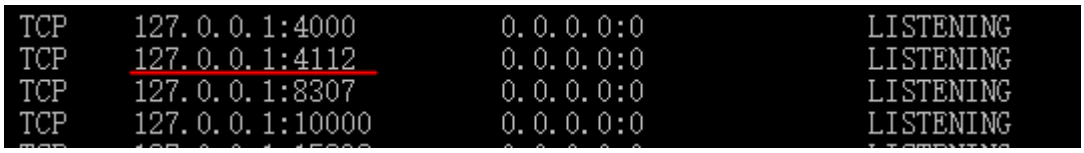
- 服务器的客户端处理子线程关键代码截图（解释总体处理逻辑，省略细节部分）

子线程中调用 process_request 接收消息，在接收到消息之后，对消息进行处理，并发送相应的请求数据，然后关闭该连接。
首先调用 process_request，用于等待用户发送地数据包。待接收到数据之后，按行读取，获取用户请求类型为[POST ,GET]中的一种，然后获取用户请求的数据文件名字和路径等信息。如果用户提交了表单，则从中获取用户输入的用户名和密码，并与预设的用户名和密码进行字符串比对。处理之后，数据都被存储在请求数据包结构体变量 request_data 中，待下一步处理。
调用 get_mime_type 函数，获取客户端请求的文件类型。调用 send_response_file 和 send_response_file 函数，给客户端发送响应数据包。
若客户端请求数据不存在，不作响应。

```
DWORD WINAPI  ThreadProcess(LPVOID PThreadData)//thread to process message
{
    size_t file_size = 0;
    char *file_type = (char*)malloc(128);
    FILE *fp;
    int bytesRecv = SOCKET_ERROR;
    //only process one message
    bytesRecv = process_request(((pThreadData)PThreadData)->AcceptSocket, ((pThreadData)PThreadData)->server_conf, &((pThreadData)PThreadData)->request_data);
    if (bytesRecv != -1)
    {//when receive data from accept socket
        if ((fp = fopen(((pThreadData)PThreadData)->request_data.file_path, "rb")) != NULL) {
            /* File Found, Now Get Some Information About File And Send It */
            get_mime_type((char*)((pThreadData)PThreadData)->request_data.file_name, file_type); // Get MIME Type
            file_size = getFileSize(fp); // Get File Size
            send_response_header(((pThreadData)PThreadData)->AcceptSocket, 200, file_type, file_size); // Send Header
            send_response_file(((pThreadData)PThreadData)->AcceptSocket, fp, file_size, ((pThreadData)PThreadData)->server_conf.max_buffer); // Send File To Client
            fclose(fp); // Close File
```

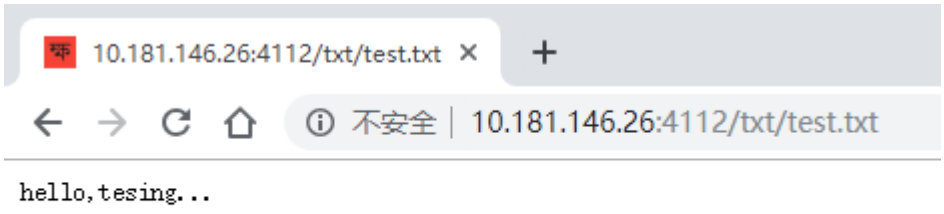
```
        }

        closesocket(((pThreadData)PThreadData)->AcceptSocket); // Close Conn
    ection
        free(PThreadData);
    }
    return 1;
}
```

- 服务器运行后，用 netstat –an 显示服务器的监听端口

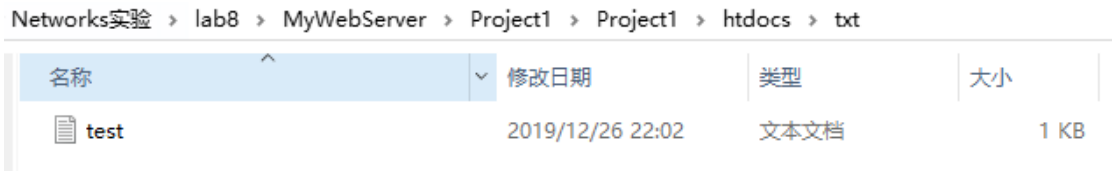


- 浏览器访问纯文本文件（.txt）时，浏览器的 URL 地址和显示内容截图。

浏览器的 URL 地址和显示内容：



服务器上文件实际存放的路径：



服务器的相关代码片段：

服务器调用 get_mime_type 函数，根据接收到的请求消息判断其类型，储存在 file_type 和 file_name 中，调用 send_response_header 和 send_response_file，生成相应的数据包并发送给客户端。

```
bytesRecv = process_request(((pThreadData)PThreadData)->AcceptSocket, ((pThreadData)PThreadData)->server_conf, &((pThreadData)PThreadData)->request_data);
    if (bytesRecv != -1)
    {//when receive data from accept socket
        if ((fp = fopen(((pThreadData)PThreadData)->request_data.file_path, "rb")) != NULL) {
            /* File Found, Now Get Some Information About File And Send It */
            get_mime_type((char*)((pThreadData)PThreadData)->request_data.file_name, file_type); // Get MIME Type
            file_size = getFileSize(fp); // Get File Size
            send_response_header(((pThreadData)PThreadData)->AcceptSocket, 200, file_type, file_size); // Send Header
            send_response_file(((pThreadData)PThreadData)->AcceptSocket, fp, file_size, ((pThreadData)PThreadData)->server_conf.max_buffer); // Send File To Client
            fclose(fp); // Close File
        }
        else {
            send_response_error(((pThreadData)PThreadData)->AcceptSocket, 404, ((pThreadData)PThreadData)->server_conf, 0); // No Debug Print
        }
        closesocket(((pThreadData)PThreadData)->AcceptSocket); // Close Connection
        free(PThreadData);
    }
```

关于 send_response_header:

```
int send_response_header(SOCKET socket, int response, char *c_type, long c_size) {
    char header[4096];

    // Add Response Code
    if (response == 200) {
        strcpy(header, "HTTP/1.1 200 OK\n");
    }

    if (response == 404) {
        strcpy(header, "HTTP/1.1 404 Not Found\n");
    }

```

```
    // Process Header
    sprintf(header, "%sServer: %s/%s (%s)\n", header, __ServerName, __Server
Version, __ServerOS);
    sprintf(header, "%sContent-Type: %s; charset=utf-8\n", header, c_type);

    sprintf(header, "%sContent-Length: %ld\n", header, c_size);
    sprintf(header, "%s\n", header);

    // Send Header
    return send(socket, header, strlen(header), 0);
}
```

关于 send_response_file:

```
int send_response_file(SOCKET socket, FILE *fp, size_t file_size, size_t max
_buffer) {
    int byteSent = SOCKET_ERROR;
    char *file_buff;

    if (max_buffer == 0 || max_buffer > file_size) {
        file_buff = (char*)malloc(file_size + 1);
        fread(file_buff, sizeof(char), file_size + 1, fp);
        byteSent = send(socket, file_buff, file_size, 0);
    }
    else {
        size_t c_position = 0;
        file_buff = (char*)malloc(max_buffer + 1);
        byteSent = 0;
        fseek(fp, 0L, SEEK_SET);

        while (c_position + max_buffer <= file_size) {
            fseek(fp, c_position, SEEK_SET);
            fread(file_buff, sizeof(char), max_buffer + 1, fp);
            byteSent += send(socket, file_buff, max_buffer, 0);
            c_position = c_position + max_buffer;
        }
        size_t left_buffer = file_size - c_position;
        if (left_buffer > 0) {
            fseek(fp, c_position, SEEK_SET);
            fread(file_buff, sizeof(char), left_buffer + 1, fp);
            if (left_buffer < max_buffer) {
                file_buff[left_buffer] = '\0';
            }
            byteSent += send(socket, file_buff, left_buffer, 0);
```

```
        }
    }

    free(file_buff);
    return byteSent;
}
```

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

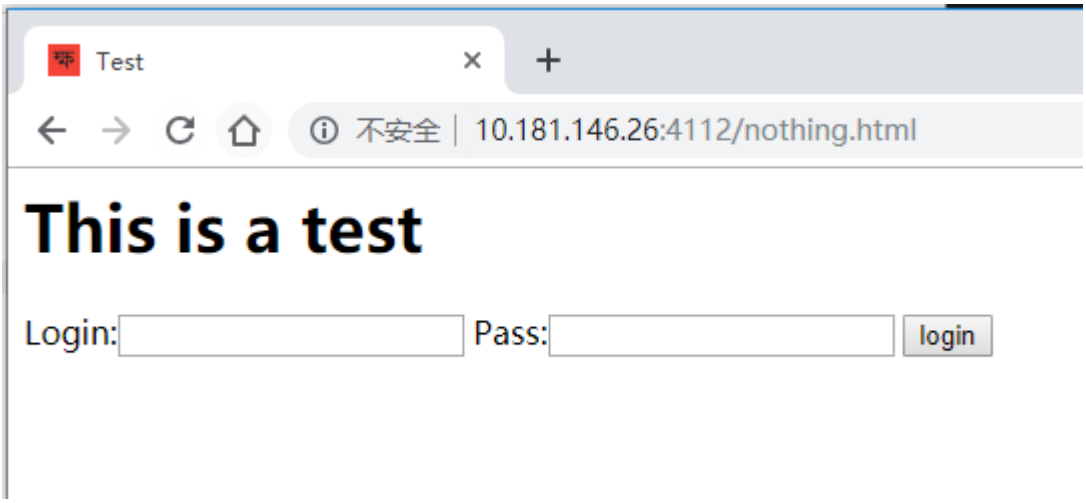
客户端发送的请求数据包：

```
Hypertext Transfer Protocol
  GET /txt/test.txt HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /txt/test.txt HTTP/1.1\r\n]
        [GET /txt/test.txt HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /txt/test.txt
      Request Version: HTTP/1.1
  Host: 10.180.184.72:4112\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (iPad; CPU OS 12_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Vers
  Accept-Language: zh-cn\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://10.180.184.72:4112/txt/test.txt]
  [HTTP request 1/1]
  [Response in frame: 47]
```

服务器发送的响应数据包：

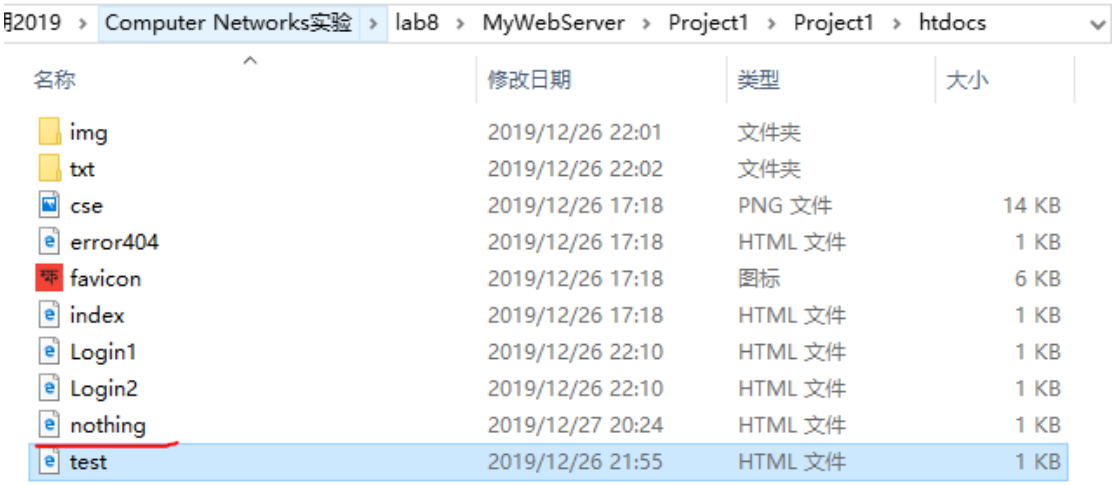
```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
        [HTTP/1.1 200 OK\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: text/plain; charset=utf-8\n
  Content-Length: 15\n
      [Content length: 15]
  \n
  [HTTP response 1/1]
  [Time since request: 0.004114000 seconds]
  [Request in frame: 23]
  File Data: 15 bytes
Line-based text data: text/plain (1 lines)
  hello,tesing...
```

- 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：

htdocs\nothing.html



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

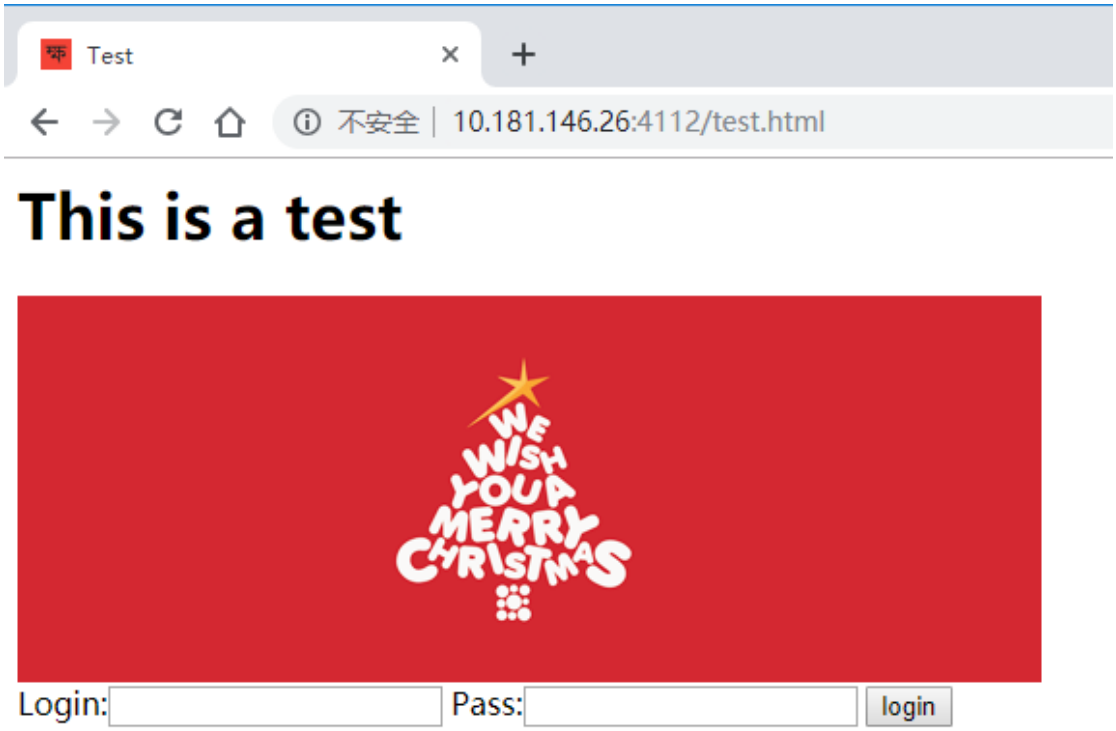
客户端请求数据包：

```
Hypertext Transfer Protocol
  GET /nothing.html HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /nothing.html HTTP/1.1\r\n]
        [GET /nothing.html HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /nothing.html
      Request Version: HTTP/1.1
  Host: 10.180.184.72:4112\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (iPad; CPU OS 12_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Vers
  Accept-Language: zh-cn\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://10.180.184.72:4112/nothing.html]
  [HTTP request 1/1]
  [Response in frame: 35]
```

服务器响应数据包：

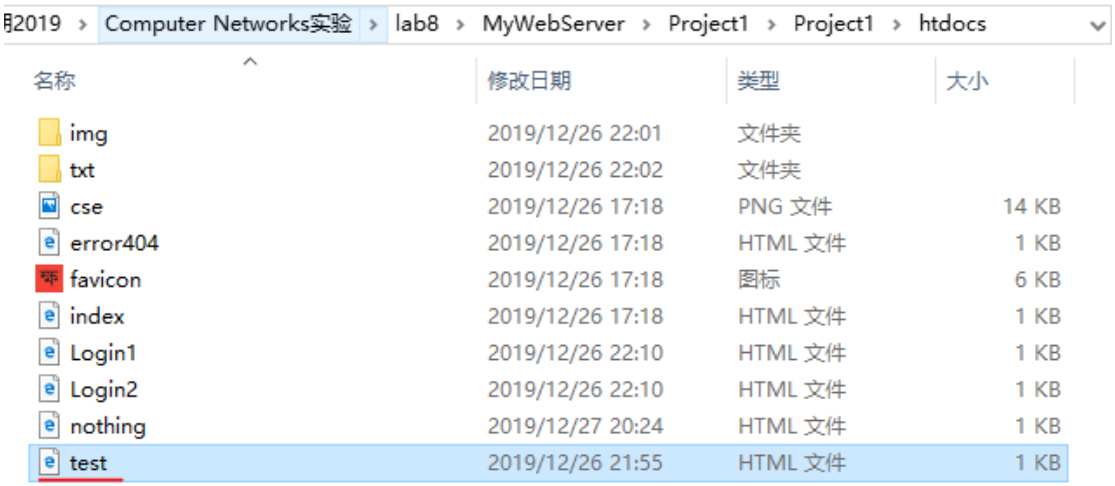
```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
        [HTTP/1.1 200 OK\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: text/html; charset=utf-8\n
  Content-Length: 243\n
      [Content length: 243]
  \n
  [HTTP response 1/1]
  [Time since request: 0.002331000 seconds]
  [Request in frame: 33]
  File Data: 243 bytes
Line-based text data: text/html (11 lines)
  <html>\r\n
  <head><title>Test</title></head>\r\n
  <body>\r\n
  <h1>This is a test</h1>\r\n
  <form action="dopost" method="POST">\r\n
      Login:<input name="login">\r\n
      Pass:<input name="pass">\r\n
      <input type="submit" value="login">\r\n
  </form>\r\n
  </body>\r\n
  </html>\r\n
```

- 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：

Htdocs\test.html



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）：

服务器发送的响应数据包：

html 部分：

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
      [HTTP/1.1 200 OK\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: text/html; charset=utf-8\n
  Content-Length: 269\n
    [Content length: 269]
  \n
  [HTTP response 1/1]
  [Time since request: 0.003049000 seconds]
  [Request in frame: 123]
  File Data: 269 bytes
Line-based text data: text/html (12 lines)
  <html>\r\n
  <head><title>Test</title></head>\r\n
  <body>\r\n
  <h1>This is a test</h1>\r\n
  <img src="img/logo.jpg">\r\n
  <form action="dopost" method="POST">\r\n
      Login:<input name="login">\r\n
      Pass:<input name="pass">\r\n
      <input type="submit" value="login">\r\n
  </form>\r\n
  </body>\r\n
  </html>\r\n
```

图片部分：

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
      [HTTP/1.1 200 OK\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: image/pjpeg; charset=utf-8\n
  Content-Length: 20338\n
    [Content length: 20338]
  \n
  [HTTP response 1/1]
  [Time since request: 0.124131000 seconds]
  [Request in frame: 122]
  File Data: 20338 bytes
Portable Network Graphics
  PNG Signature: 89504e470d0a1a0a
  Image Header (IHDR)
  Image gamma (gAMA)
  Standard RGB colour space (sRGB)
  Primary chromaticities and white point (cHRM)
  Background colour (bKGD)
  Physical pixel dimensions (pHYs)
  Image data chunk (IDAT)
  Image Trailer (IEND)
```

客户端接受的数据包：

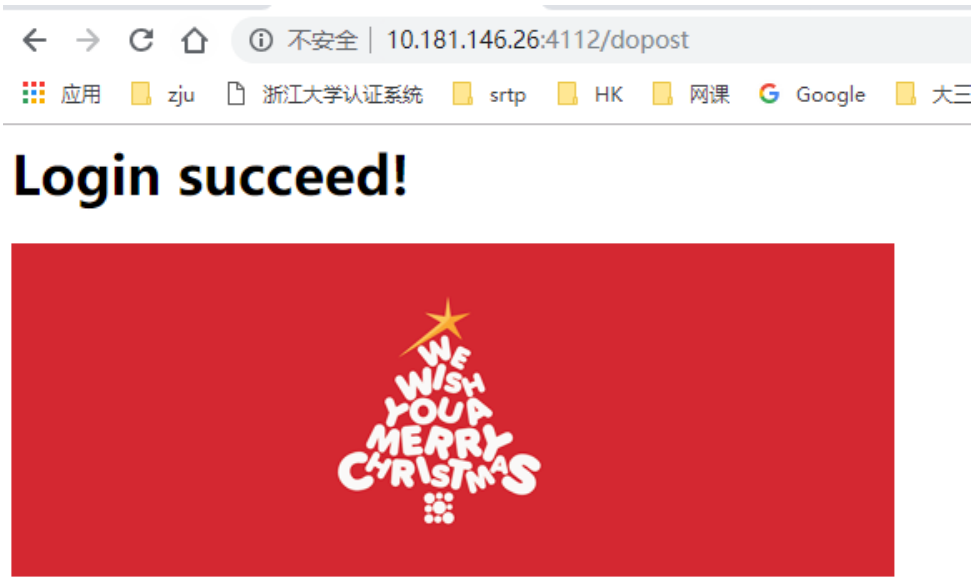
Html 部分：

```
Hypertext Transfer Protocol
  GET /test.html HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /test.html HTTP/1.1\r\n]
      [GET /test.html HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: GET
    Request URI: /test.html
    Request Version: HTTP/1.1
  Host: 10.180.184.72:4112\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (iPad; CPU OS 12_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Vers
  Accept-Language: zh-cn\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://10.180.184.72:4112/test.html]
  [HTTP request 1/1]
  [Response in frame: 344]
```

图片部分：

```
Hypertext Transfer Protocol
  GET /img/logo.jpg HTTP/1.1\r\n
  Host: 10.180.184.72:4112\r\n
  Connection: keep-alive\r\n
  Accept: image/png,image/svg+xml,image/*;q=0.8,video/*;q=0.8,*/*;q=0.5\r\n
  User-Agent: Mozilla/5.0 (iPad; CPU OS 12_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Versi
  Accept-Language: zh-cn\r\n
  Referer: http://10.180.184.72:4112/test.html\r\n
  Accept-Encoding: gzip, deflate\r\n
  \r\n
  [Full request URI: http://10.180.184.72:4112/img/logo.jpg]
  [HTTP request 1/1]
  [Response in frame: 404]
```

- 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：通过将字符串分段，并获取一定位置处的字符串，得到用户输入的用户名和密码，与正确输入进行字符串比对，若正确，则将文件名修改为登录成功的文件名，否则修改为登录失败的文件名。

```
receive_array = strtok(buffer_data, "\n");

//to get the password
for (int i = 0; i < 14; i++)
{
    password = strtok(NULL, "\n");
}
if (strcmp(file_path, "htdocs\\dopost") == 0)//form data
{
    char CorrectPassword[27] = "login=3170104112&pass=4112";
    char* UserPassword = (char*)malloc(27);
    memcpy(UserPassword, password, 27);
    UserPassword[26] = '\0';
    if (strcmp(CorrectPassword, UserPassword) == 0)
    {
        memcpy(request_data->file_name, "htdocs\\Login1.html", 19);
        memcpy(request_data->file_path, "htdocs\\Login1.html", 19);
        memcpy(request_data->requested_uri, "htdocs\\Login1.html", 19);
    }
    else
    {
        memcpy(request_data->file_name, "htdocs\\Login2.html", 19);
```

```
23.         memcpy(request_data->file_path, "htdocs\\Login2.html", 19);
24.         memcpy(request_data->requested_uri, "htdocs\\Login2.html", 19);
25.     }
26. }
```

Wireshark 抓取的数据包截图（HTTP 协议部分）

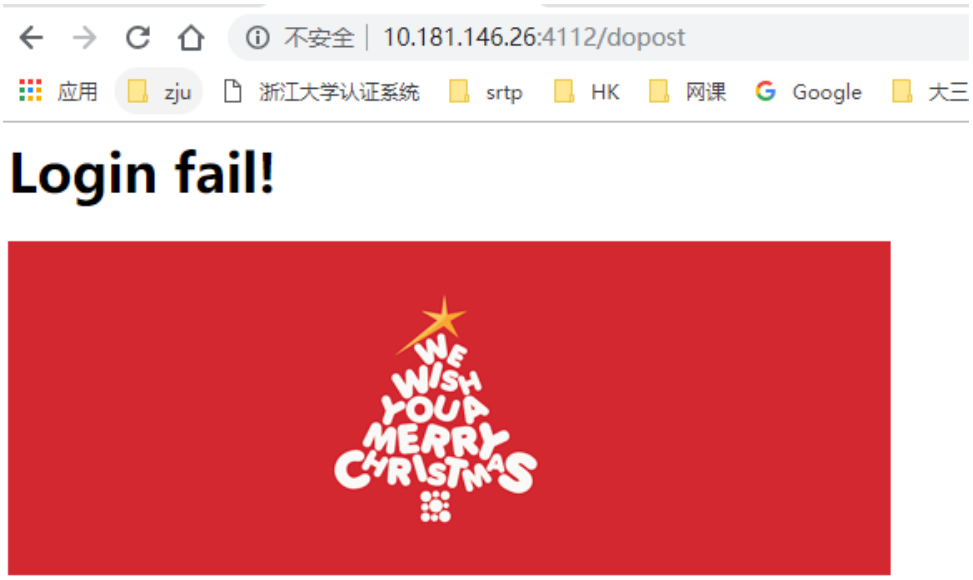
客户端表单数据包：

```
Hypertext Transfer Protocol
  POST /dopost HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): POST /dopost HTTP/1.1\r\n]
      [POST /dopost HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: POST
    Request URI: /dopost
    Request Version: HTTP/1.1
  Host: 10.181.146.26:4112\r\n
  Connection: keep-alive\r\n
  Content-Length: 26\r\n
  Cache-Control: max-age=0\r\n
  Origin: http://10.181.146.26:4112\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  User-Agent: Mozilla/5.0 (Linux; U; Android 9; en-us; PCAM00 Build/PKQ1.190519.001) AppleWebKit/537.3
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n
  Referer: http://10.181.146.26:4112/test.html\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US\r\n
  \r\n
  [Full request URI: http://10.181.146.26:4112/dopost]
  [HTTP request 1/1]
  [Response in frame: 34]
  File Data: 26 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "login" = "3170104112"
    Key: login
    Value: 3170104112
  Form item: "pass" = "4112"
    Key: pass
    Value: 4112
```

服务器响应数据包：

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
      [HTTP/1.1 200 OK\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: text/html; charset=utf-8\n
  Content-Length: 128\n
  \n
  [HTTP response 1/1]
  [Time since request: 0.004027000 seconds]
  [Request in frame: 32]
  File Data: 128 bytes
Line-based text data: text/html (7 lines)
  <html>\r\n
  <head><title>Login succeed</title></head>\r\n
  <body>\r\n
  <h1>Login succeed!</h1>\r\n
  <img src="img/logo.jpg">\r\n
  </body>\r\n
  </html>\r\n
```

- 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）

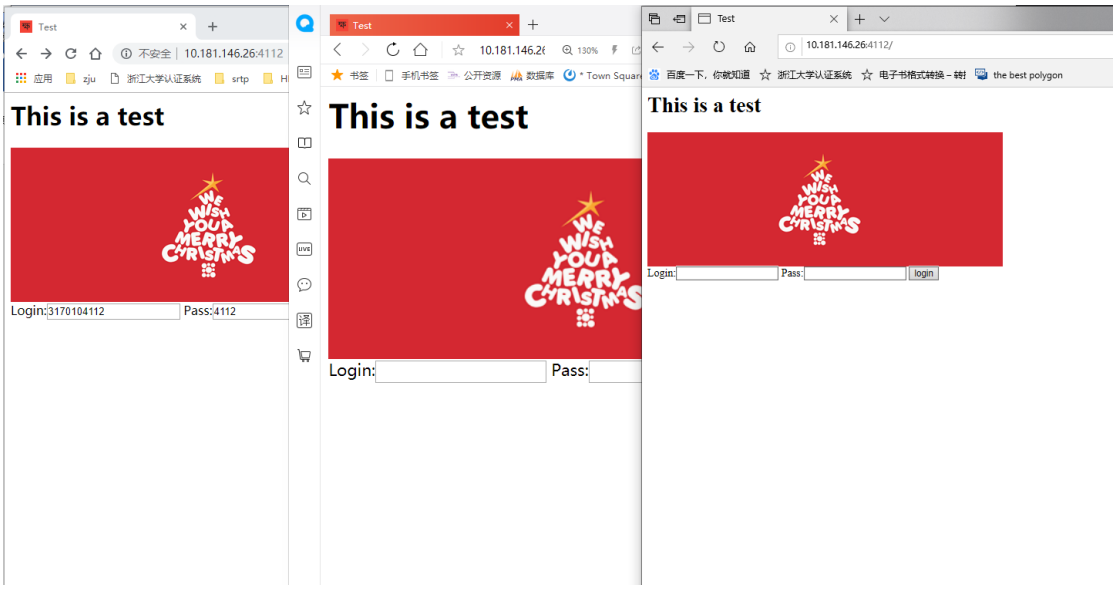
客户端表单数据包：

```
Hypertext Transfer Protocol
  POST /dopost HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): POST /dopost HTTP/1.1\r\n]
      [POST /dopost HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: POST
    Request URI: /dopost
    Request Version: HTTP/1.1
  Host: 10.181.146.26:4112\r\n
  Connection: keep-alive\r\n
  Content-Length: 26\r\n
  Cache-Control: max-age=0\r\n
  Origin: http://10.181.146.26:4112\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  User-Agent: Mozilla/5.0 (Linux; U; Android 9; en-us; PCAM00 Build/PKQ1.190519.001) AppleWebKit/537.36 (K
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n
  Referer: http://10.181.146.26:4112/test.html\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US\r\n
  \r\n
  [Full request URI: http://10.181.146.26:4112/dopost]
  [HTTP request 1/1]
  [Response in frame: 128]
  File Data: 26 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "login" = "3170104112"
    Key: login
    Value: 3170104112
  Form item: "pass" = "4111"
    Key: pass
    Value: 4111
```

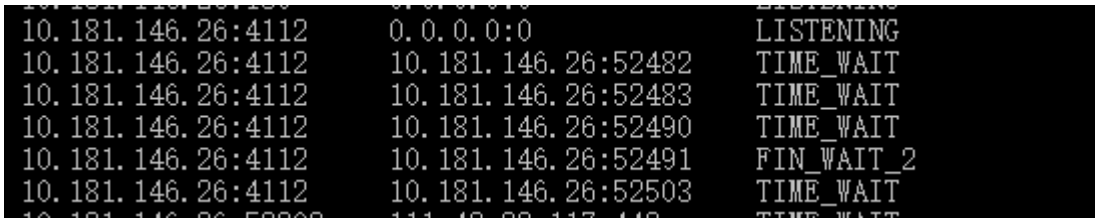
服务器响应数据包：

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\n]
      [HTTP/1.1 200 OK\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Server: DQ's server/0.1.1 (Win32)\n
  Content-Type: text/html; charset=utf-8\n
  Content-Length: 122\n
  \n
  [HTTP response 1/1]
  [Time since request: 0.001169000 seconds]
  [Request in frame: 126]
  File Data: 122 bytes
Line-based text data: text/html (7 lines)
  <html>\r\n
  <head><title>Login Fail</title></head>\r\n
  <body>\r\n
  <h1>Login fail!</h1>\r\n
  <img src="img/logo.jpg">\r\n
  </body>\r\n
  </html>\r\n
```

- 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）



六、 实验结果与分析

根据你编写的程序运行效果，分别解答以下问题（看完请删除本句）：

- HTTP 协议是怎样对头部和体部进行分隔的？

  通过一个空行，告诉服务器请求头部到此为止。

- 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？

  根据文件拓展名判断，在 get_mime_type 函数中对传入的文件名字后缀进行判断。

- HTTP 协议的头部是不是一定是文本格式？体部呢？

  是的。是的。

- 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？

体部。用&符号连起来。

七、 讨论、心得

在实验 7 的基础上，我进一步了解了 socket 编程的内容，并且阅读了 http 协议详细的协议内容。在探索中遇到了很多困难，包括编程中的困难和对协议内容的理解，在不断尝试中我最终能比较好地理解这些知识，并完成实验内容。