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：	胡若凡
学　　院：	计算机学院
系：	计算机科学与技术学院
专　　业：	计算机科学与技术
学　　号：	3200102312
指导教师：	张泉方

2022 年 11 月 15 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**

以下实验记录均需结合屏幕截图（截取源代码或运行结果），进行文字标注（看完请删除本句）。

- **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 (1) {
            cli_addr_len = sizeof(sock_Http_addr);
            cli_sock = accept(socket_listen, (struct sockaddr
*)&sock_Http_addr, &cli_addr_len);
            if(cli_sock == INVALID_SOCKET)
            {
                printf("accept() failed!\n");
                closesocket(socket_listen);//关闭套接字
                WSACleanup();
                return 0;
            }
            HANDLE server_thread;
```

```
            CreateThread(NULL, 0, thread_handle, &cli_sock, 0, 0);
            //创建在调用进程的虚拟地址空间内执行的线程。sServer 是传递给线程的
变量的指针，handlemsg 是执行代码
        }
```

解释：这里同 socket 实验一样，主线程中通过 while 循环，不断去 accept。对于单此 accept，如果接收成功，那么就创建一个子线程，在子线程中去

- **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）**

```
    while (1)//对于该请求也会有不断变化
    {
        char buf[1024*64],method[1024],
             url[1024],version[1024],name[1024];
        if (recv(tcp,buf,1024*64,0<=0)
        {
            cout<<"Recieved failed"<<endl;
        }
        else{
            cout<<endl;
            cout<<endl;
            cout<<"The recived packet we list here:"<<endl;
            cout<<buf<<endl;
            sscanf(buf,"%s%s%s",method,url,version);
            cout<<"method = "<<method<<endl;
            cout<<"url = "<<url<<endl;
            cout<<"version = "<<version<<endl;
            //打印出 http 里的三个部分
            if (stricmp(method,"GET")==0)
            {
            }
            else
            if (stricmp(method, "POST") == 0)
            {
            }
        }
```

解释：对于子线程中，也要用一个 while 循环去不断接收此次的请求。首先会判断 recv 是否成功接收，如果成功接收，那么进入处理。处理中会对 method 进行解析，然后分别进入 GET 与 POST 两种不同请求的处理部分。

下面先给出 GET 请求的处理：

```
    if (stricmp(method,"GET")==0)
    {
```

```
        cout<<"we handle GET now"<<endl;
        char type[32];
        strcpy(name, "../src");//先进行一个赋值，src 是我存放文件的地方
        //strstr 函数用来对字符数组进行类似的 find 操作
        //这里的坑是，寻找对应的一定要加上.img 类型，原因在于 noimg.html 的名称
        if (strstr(url,".txt")!=NULL){
            strcpy(type,"text/plain");
            strcat(name,"/txt");
        }
        else if (strstr(url,".html")!=NULL){
            strcat(name, "/html");//继续对对应的文件夹进行寻找
            strcpy(type, "text/html");//赋上对应的类型
        }
        else if (strstr(url,".img")!=NULL){
            strcpy(type,"image/jpeg");
        }
        strcat(name,url);//得到文件要去哪里寻找
        cout<<"we are now finding the file in "<<name<<endl;
        FILE *f=fopen(name,"rb");
        if (f!=NULL)//如果找得到
        {
            cout<<"we have get the file"<<endl;
            fseek(f,0,SEEK_END);
            char len[10];
            long filelen=ftell(f);
            _ltoa(filelen,len,10);
            char *content=new char[filelen];
            fseek(f,0,SEEK_SET);//回到头部，也可以用 rewind
            fread(content,sizeof(char),filelen,f);
            if(content == NULL) {
                printf("[file get error]\n");
                break;
            }
            char sendbuf[1024*64]="HTTP/1.1 200 OK\r\n";
            strcat(sendbuf, "Connection: keep-alive\r\n");
            strcat(sendbuf,"Content-Length:"); strcat(sendbuf, len);
strcat(sendbuf,"\r\n");
            strcat(buf, "Server: csr_http1.1\n");
            if (strstr(url, ".img")) strcat(buf, "Accept-Ranges:
bytes\r\n");
            strcat(sendbuf,"Content-Type"); strcat(sendbuf,type);
strcat(sendbuf,"\r\n");
            strcat(sendbuf,"\r\n");
            int headlen=strlen(sendbuf);
```

```
            memcpy(sendbuf+strlen(sendbuf), content, filelen);
            if (send(tcp,sendbuf,filelen+headlen,0))
                cout<<"Send successfully"<<endl;
            else cout<<"Send failed"<<endl;
        }
        else{
            cout << "No such file existed: "<< name << endl;
            char sendbuf[1024*64] = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html;charset=utf-8\r\nContent-Length: 84\r\n\r\n";
            strcat(sendbuf, "<html><body><h1>404 Not Found</h1><p>From
server: URL is wrong.</p></body></html>\r\n");
            if (!send(tcp, sendbuf, strlen(sendbuf), 0))
                cout<<"Couldn't find message send error"<<endl;
        }
        fclose(f);
    }
```

GET 的请求中，首先要处理的文件寻址，对于这次实验中给定的三种后缀名进行寻址（一定要是后缀名），附上文件相对于.exe 程序的路径地址，然后去寻找文件。如果找到了文件，那么就打包发送（尤其是 img 的发送要有 Accept-Range）。如果没有找到就直接发送打包一个错误的头和 html 内容进行发送。

```
    if (stricmp(method, "POST") == 0)
    {
        cout<<"we handle POST now"<<endl;
        if (stricmp(url, "/dopost") != 0) {
            cout << "Dopost!错误" << endl;
            char sendbuf[1024*64] = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html;charset=utf-8\r\nContent-Length: 84\r\n\r\n";
            strcat(sendbuf, "<html><body><h1>404 Not Found</h1><p>From
server: URL is wrong.</p></body></html>\r\n");
            if (!send(tcp, sendbuf, strlen(sendbuf), 0)){
                cout<<"Couldn't find message send error"<<endl;
            }
        }
        else{
            char* login=strstr(buf,"login=");
            //回到头部，接着要对 login 进行内容的查找，格式为 login=&password=
            int cnt=0;
            while (login[cnt]!='\0'){
                if (login[cnt]=='&') break;
                cnt++;
            }
```

```
            char username[30],password[30];
            strncpy(username,login+strlen("login="),cnt-strlen("login=")
);//截取用户名
            strncpy(password,login+cnt+strlen("&pass="),strlen(login)-cn
t); //截取密码
            cout<<username<<endl;
            cout<<password<<endl;
            char sendbuf[1024*64]= "HTTP/1.1 200 OK\r\n";
            strcat(sendbuf, "Connection: keep-alive\r\n");
            strcat(sendbuf, "Server: csr_http1.1\r\n");
            strcat(sendbuf, "Content-Type: text/html\r\n");
            strcat(sendbuf, "Content-Length: ");
            if (strcmp(username, "3200102312") == 0 && strcmp(password,
"2312") == 0)
            {           //登录成功
                char *msg = (char*)"<html><body>Login
Success!Congratulations!</body></html>";
                char len[10];
                itoa(strlen(msg), len, 10);
                strcat(sendbuf, len);
                strcat(sendbuf, "\r\n\r\n");
                strcat(sendbuf, msg);
                if(!send(tcp, sendbuf, strlen(sendbuf), 0)) {
                    printf("[send error]\n");
                }
            }
            else {
                char *msg = (char*)"<html><body>Login
Failed!</body></html>";
                char len[10];
                itoa(strlen(msg), len, 10);
                strcat(sendbuf, len);
                strcat(sendbuf, "\r\n\r\n");
                strcat(sendbuf, msg);
                if(!send(tcp, sendbuf, strlen(sendbuf), 0)) {
                    printf("[send error]\n");
                }
            }
        }
    }
```

POST 的处理和之前的十分类似，这里先判断请求类型是 POST，然后就开始对登录进行处理。由于在两个登录的 html 文件中 Login:<input name="login">Pass:<input name="pass"> 着两句，因此要先提取出用户名和密码。对用户名密码进行判断后，分别组装不一样的报文

进行发送即可。

- **服务器运行后，用 netstat –an 显示服务器的监听端口**

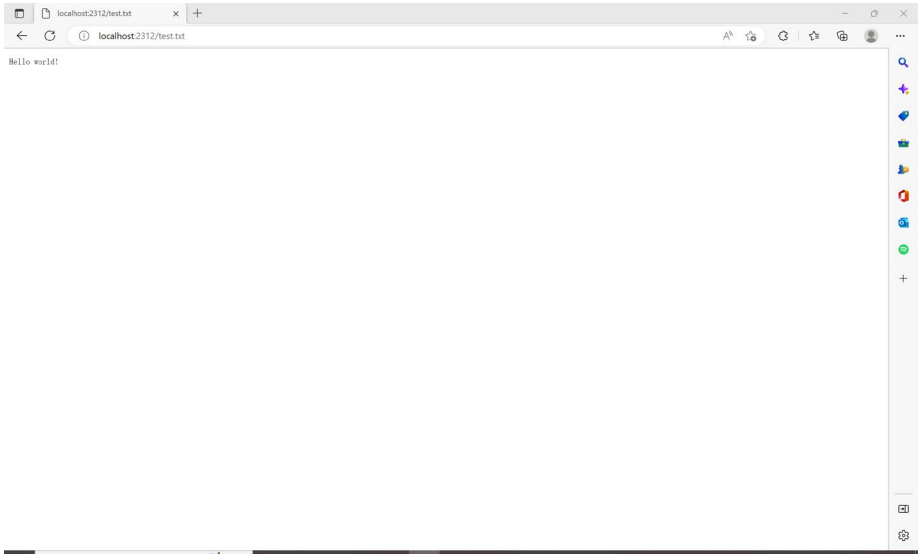
  如图所示，最后一行的 2312 即为我的服务器监听端口，这里是我的学号后 4 位。

```
C:\Users\lenovo>netstat -an

活动连接

  协议   本地地址          外部地址        状态
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:902            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:912            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:2312           0.0.0.0:0              LISTENING
```

- **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。**



服务器上文件实际存放的路径：

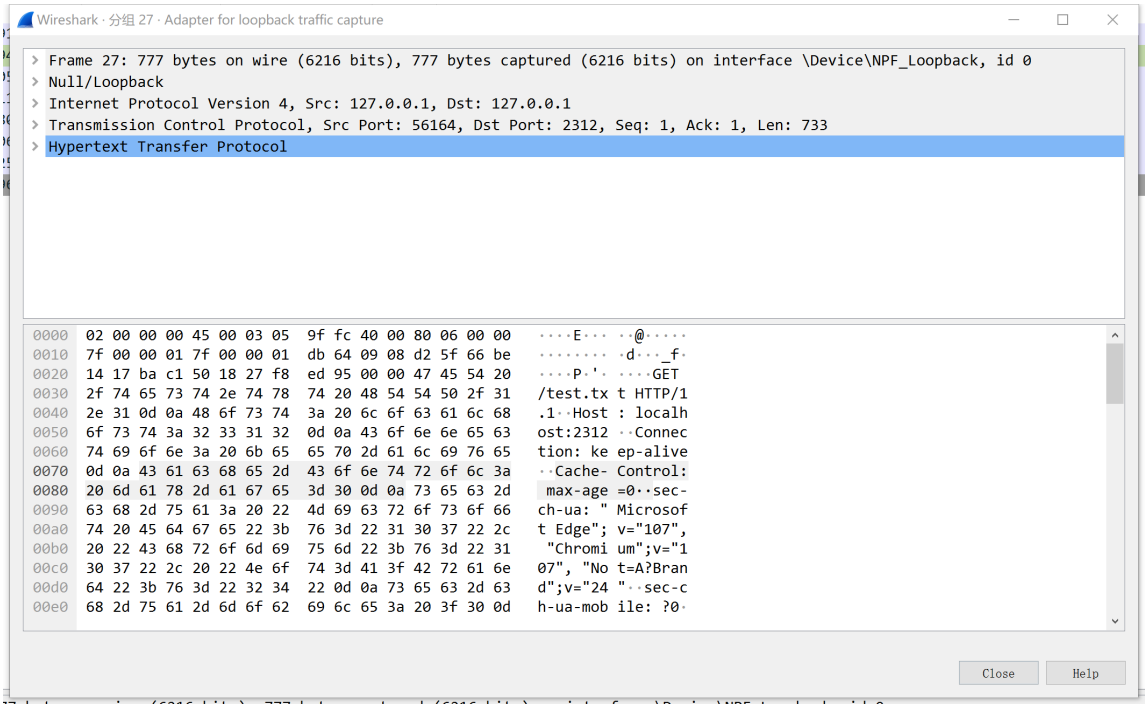
C:\Users\lenovo\Desktop\Homework-Web\src\txt

服务器的相关代码片段：

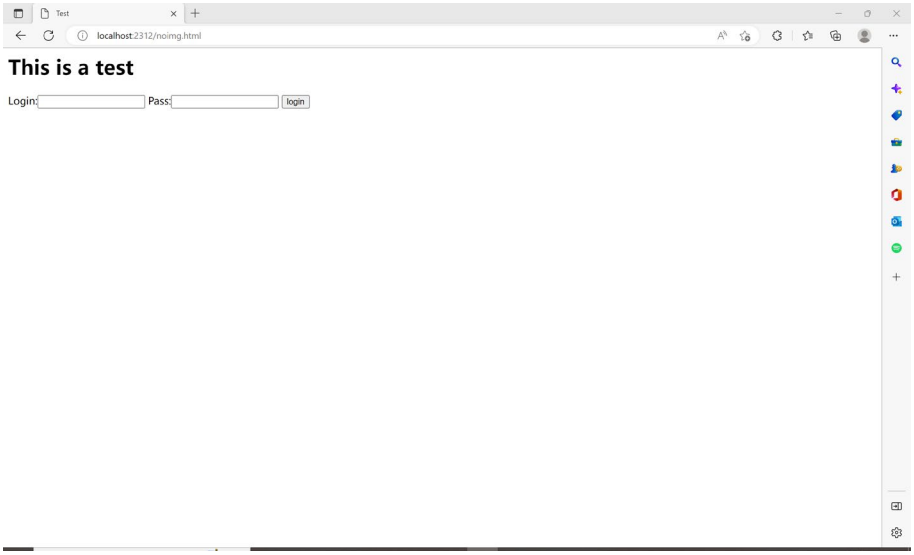
```
if (strstr(url,".txt")!=NULL){
    strcpy(type,"text/plain");
    strcat(name,"/txt");
}
```

具体分析已在上方完成，这里仅显示判断语句。

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



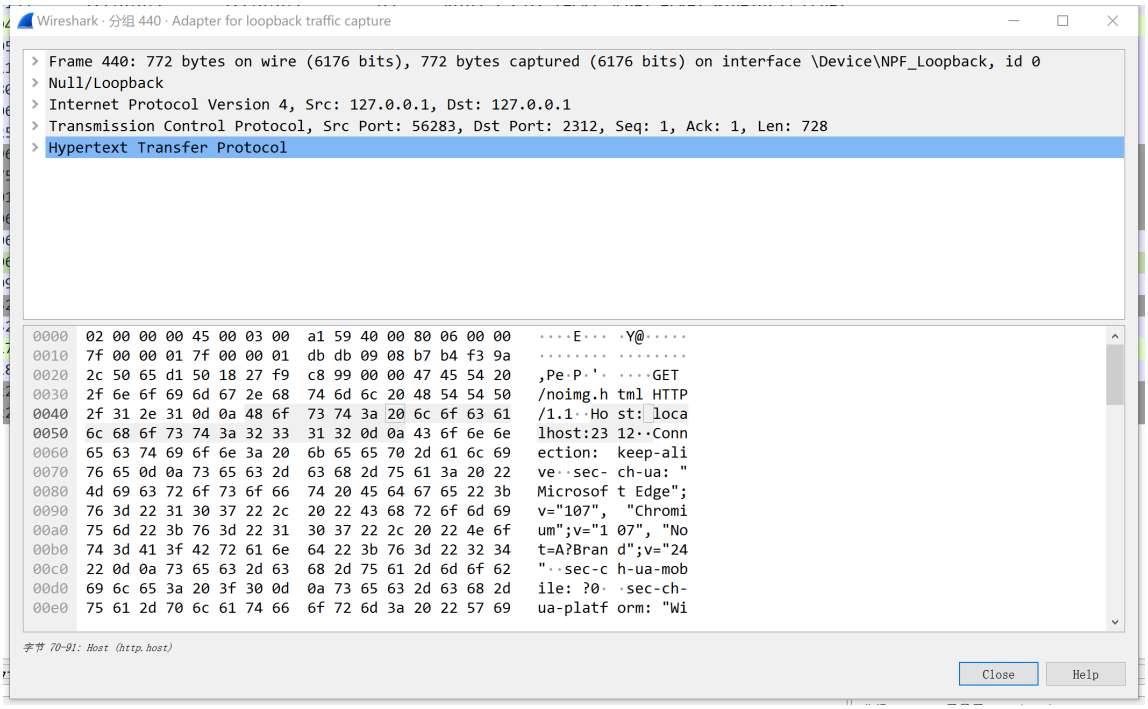
- **浏览器访问只包含文本的 HTML 文件时，浏览器的 URL 地址和显示内容截图。**



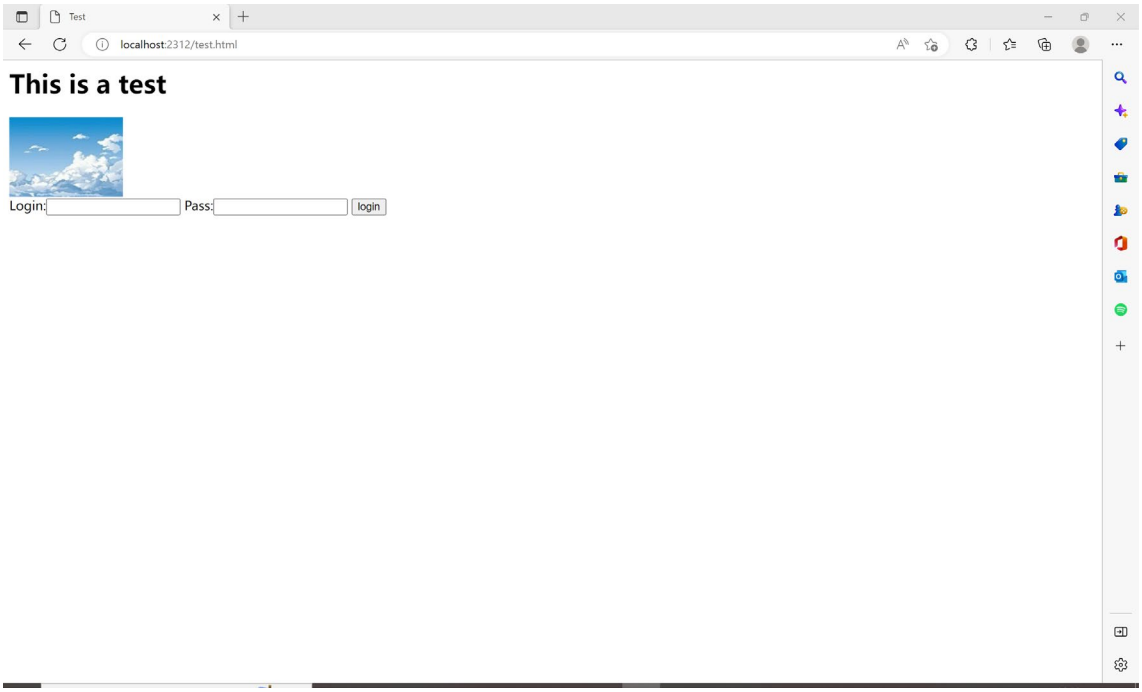
服务器文件实际存放的路径：

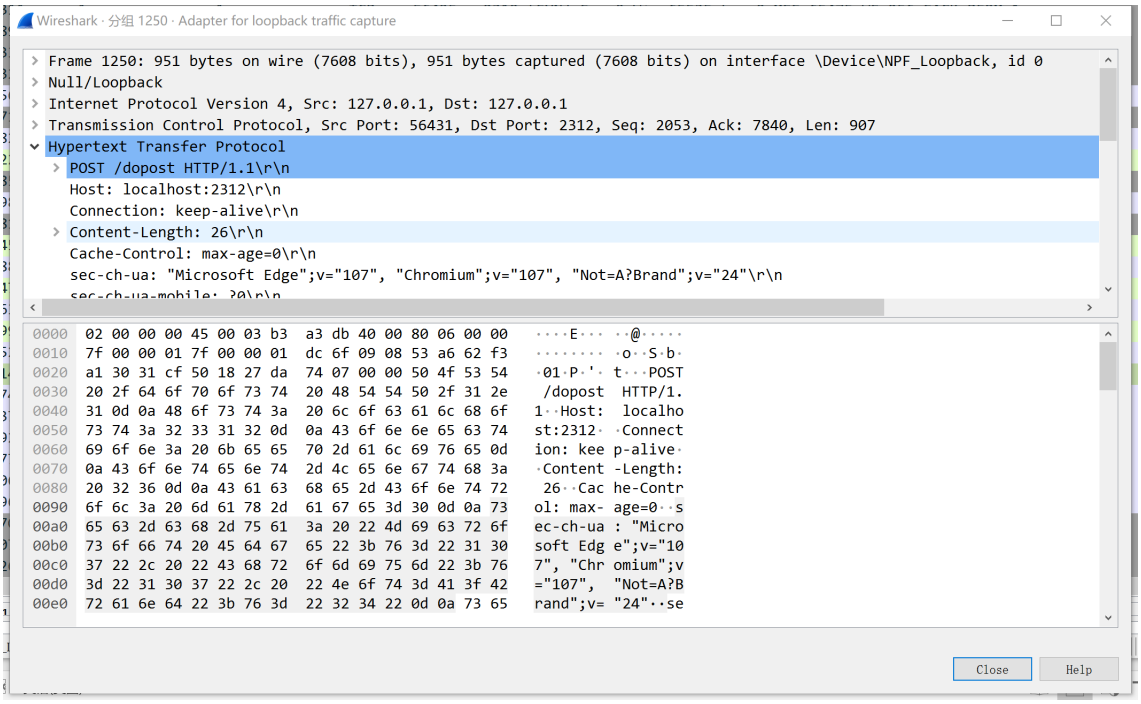
C:\Users\lenovo\Desktop\Homework-Web\src\html\noimg.html

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



- **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。**



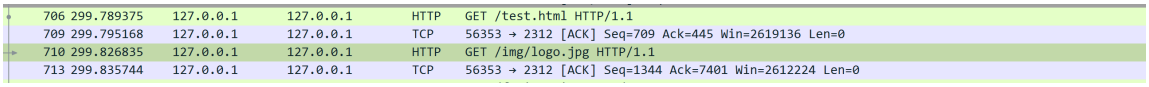


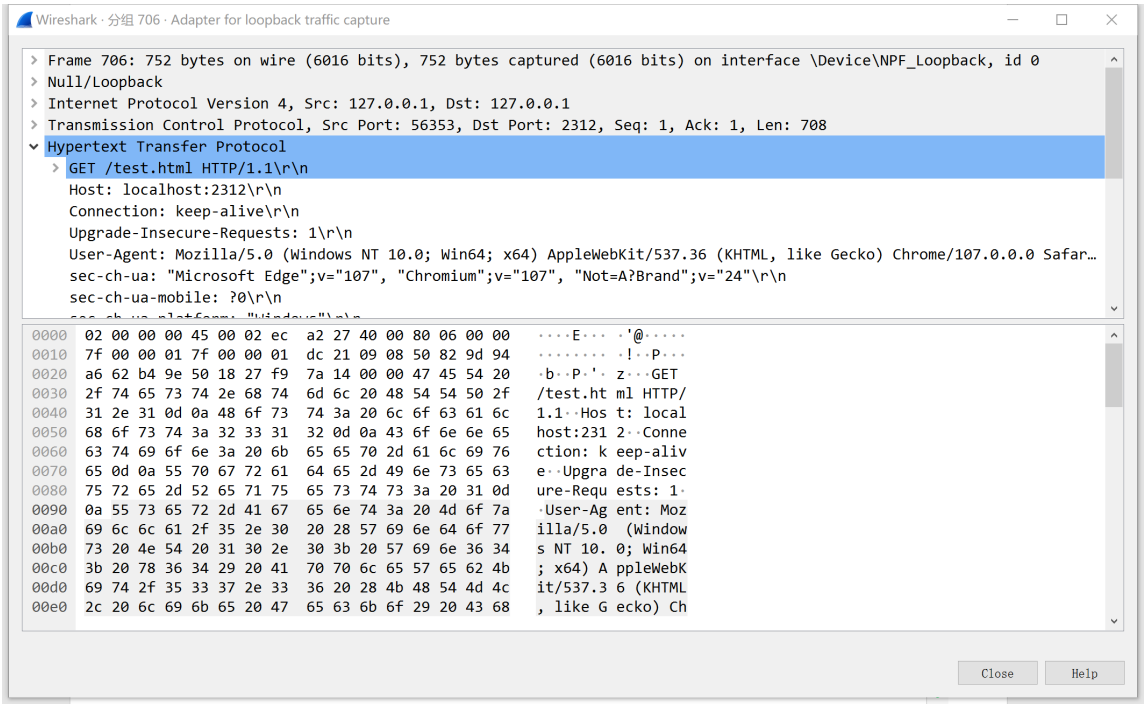
服务器上文件实际存放的路径：

C:\Users\lenovo\Desktop\Homework-Web\src\html\test.html

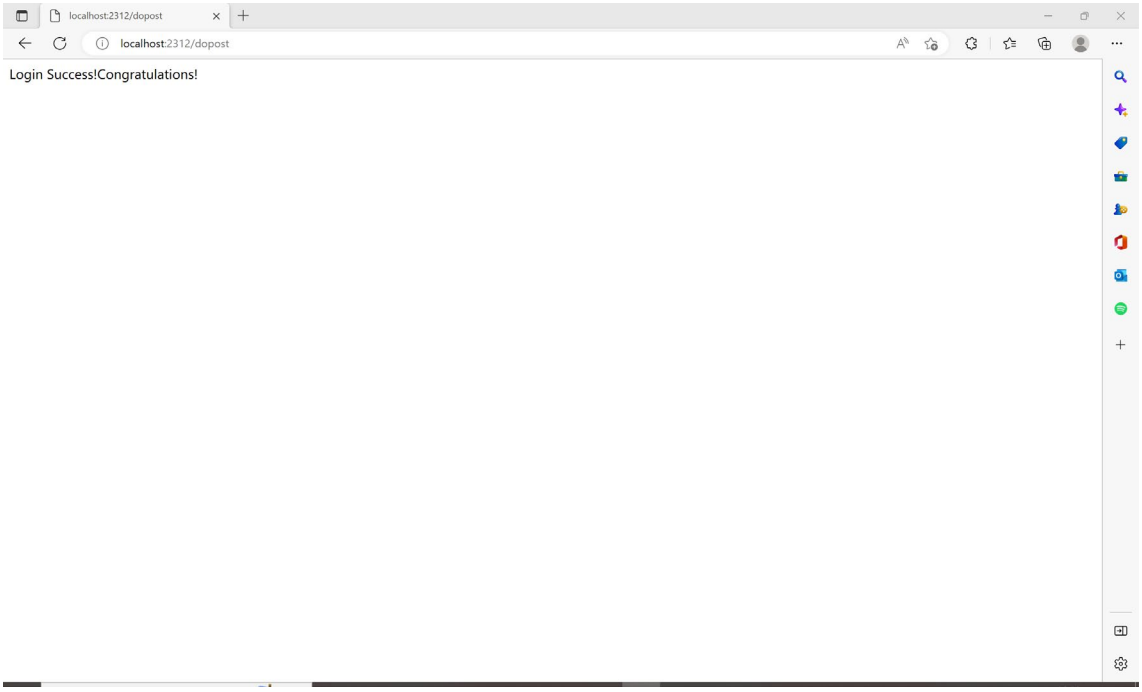
C:\Users\lenovo\Desktop\Homework-Web\src\img\logo.jpg

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）：





- 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：

```
        char* login=strstr(buf,"login=");
        cout<<login<<endl;
        int cnt=0;
        while (login[cnt]!='\0'){
```

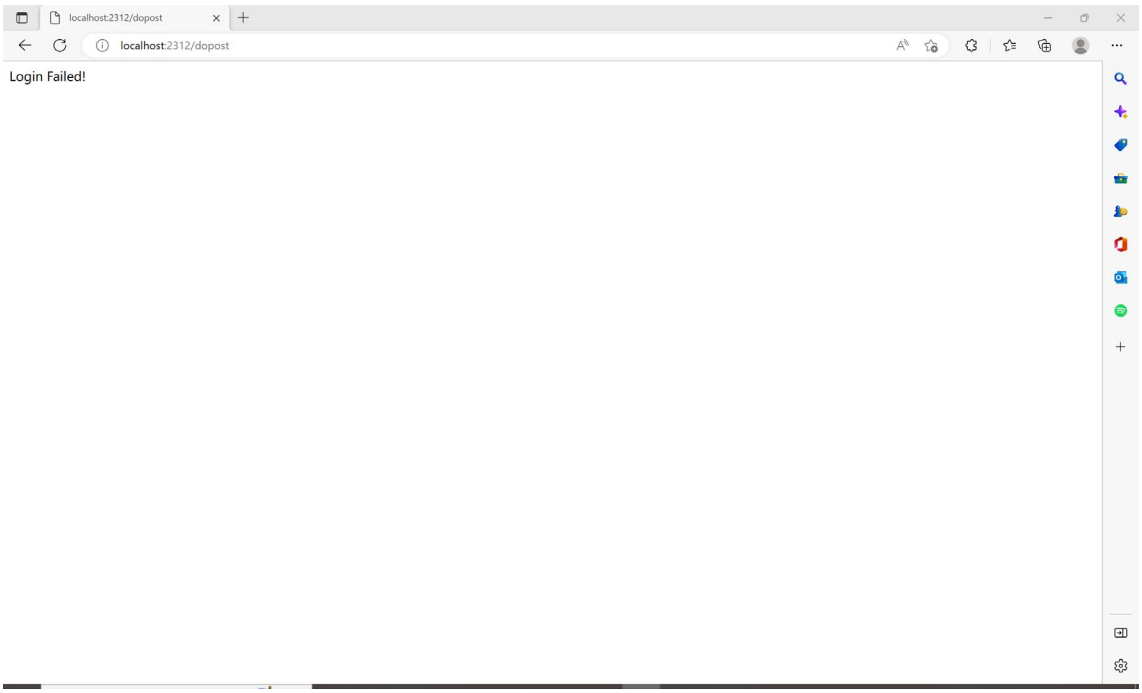
```
                if (login[cnt]=='&') break;
                cnt++;
            }
            char username[30],password[30];
            strncpy(username,login+strlen("login="),cnt-strlen("login=")
);
            strncpy(password,login+cnt+strlen("&pass="),strlen(login)-cn
t);
            cout<<username<<endl;
            cout<<password<<endl;
            char sendbuf[1024*64]= "HTTP/1.1 200 OK\r\n";
            strcat(sendbuf, "Connection: keep-alive\r\n");
            strcat(sendbuf, "Server: csr_http1.1\r\n");
            strcat(sendbuf, "Content-Type: text/html\r\n");
            strcat(sendbuf, "Content-Length: ");
            if (strcmp(username, "3200102312") == 0 && strcmp(password,
"2312") == 0)
            {           //登录成功
                char *msg = (char*)"<html><body>Login
Success!Congratulations!</body></html>";
                char len[5];
                itoa(strlen(msg), len, 10);
                strcat(sendbuf, len);
                strcat(sendbuf, "\r\n\r\n");
                strcat(sendbuf, msg);
                if(!send(tcp, sendbuf, strlen(sendbuf), 0)) {
                    printf("[send error]\n");
                }
            }
```

分析：先提取用户名和密码，然后成功后设置一段成功的 html，并进行封装

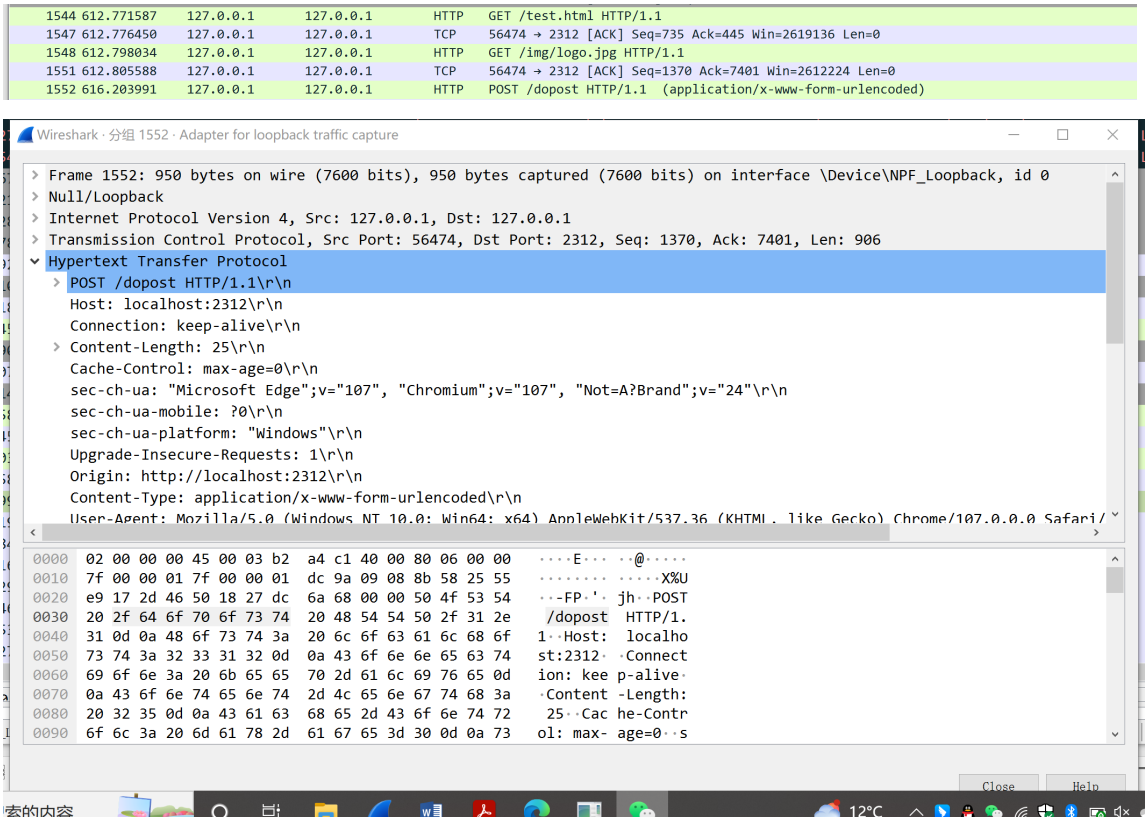
Wireshark 抓取的数据包截图（HTTP 协议部分）

1209	495.895885	127.0.0.1	127.0.0.1	TCP	56431 → 2312 [ACK] Seq=710 Ack=440 Win=2619136 Len=0
1234	508.832471	127.0.0.1	127.0.0.1	HTTP	GET /test.html HTTP/1.1
1237	508.835636	127.0.0.1	127.0.0.1	TCP	56431 → 2312 [ACK] Seq=1418 Ack=884 Win=2618880 Len=0
1238	508.857995	127.0.0.1	127.0.0.1	HTTP	GET /img/logo.jpg HTTP/1.1
1241	508.859529	127.0.0.1	127.0.0.1	TCP	56431 → 2312 [ACK] Seq=2053 Ack=7840 Win=2611712 Len=0
1250	512.552148	127.0.0.1	127.0.0.1	HTTP	POST /dopost HTTP/1.1 (application/x-www-form-urlencoded)

- **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。**

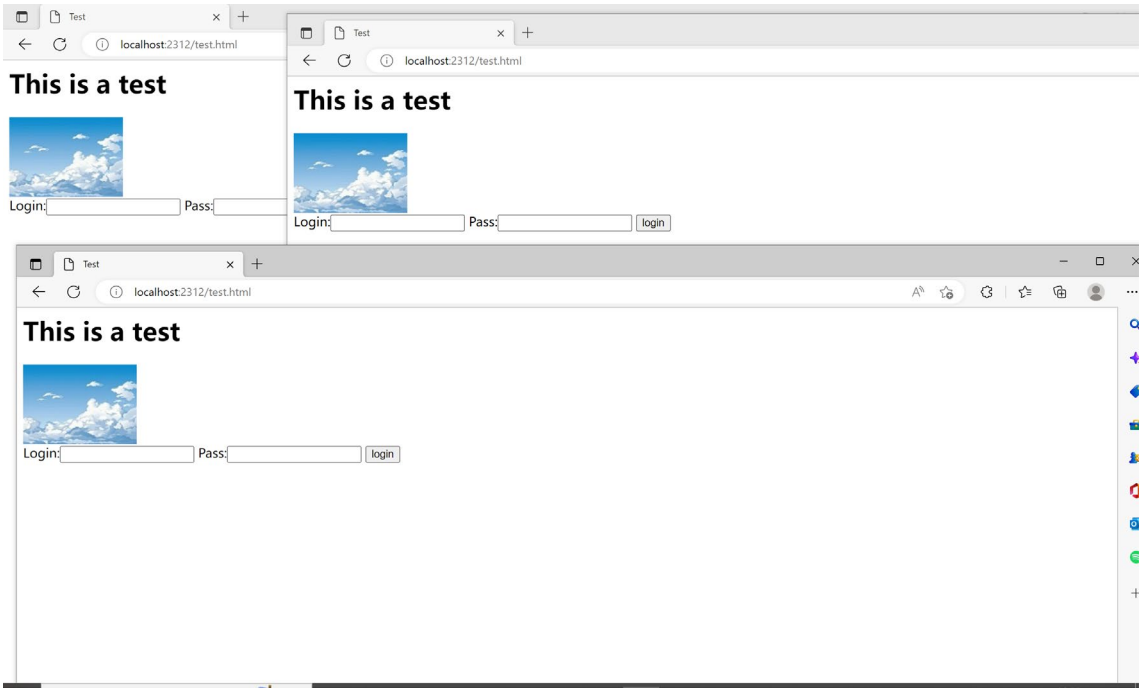


- **Wireshark 抓取的数据包截图（HTTP 协议部分）**



- **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    127.0.0.1:2312         127.0.0.1:61950        CLOSE_WAIT
TCP    127.0.0.1:2312         127.0.0.1:61951        ESTABLISHED
TCP    127.0.0.1:2312         127.0.0.1:61952        ESTABLISHED
```

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？**

  答：有一行空行来分隔头部和体部。

- **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**

  答：浏览器是根据头部的 Content-Type 字段来判断文件类型的。本次实验中处理的有三类，分别是 text/html ： HTML 格式，text/plain，文本格式，image/jpeg ： jpg 图片格式。

- **HTTP 协议的头部是不是一定是文本格式？体部呢？**

  答：协议的头部是文本格式，体部除了文本格式外，还可以是字节流的方式，比如本次实验中就对图片进行了传输。

- **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？**

  答：放在体部，如“login=123&pass=12345”，两个字段通过&符号连接。

七、 讨论、心得

心得：在这次实验中，我尝试换了 winsock 的编程模式。整体上来看，各种操作和语句和之前我在 linux 环境下的 socket 编程是十分类似的，只是偶尔有一两个语句有少许区别，总体来看也不是不能接受，就是抓包的时候会碰到很多问题。

从实验设计上，本次实验承接之前的 socket 编程，大致思路都是一样的，多的只是发送的内容，牵涉到了报文的格式。这一点只要在网上找一个具体的网页详细介绍一下请求报文格式和响应报文格式就好，理解行、头部、包体都有什么又怎样隔开。理论理解不成困难，然后下面的这个网页是我进行理论学习的网站，感觉整体很清楚 https://cloud.tencent.com/developer/article/1953222

但是实践起来却不是那么容易的，我这里列举几个我碰到的问题：（1）在进行类型判断的时候，一开始我是去找有无 html,img 这样的，但是由于一些名称带 img,所以 debug 了很久，这让我更加明确之后写代码一定要去找.img 的类型，加上.会好很多（2）对于发送的响应报文，在响应头部里到底要放多少东西，这里也很重要，一开始我只是简单放了 length 等，后来逐渐发现一些东西不放会出问题，所以一点点加起来了（3）对于图片，这个和发送文本不太一样，是字节流的模式，因为这个是在对 test.html 测试的时候才发现的，又在网上查找了很久才知道要对长度进行处理。

通过这次实验，我对 socket 实验更清楚了，对于抓包也更明白了。