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：　计算机学院

专　　业：　计算机科学与技术

学　　号：

指导教师：　黄正谦

2022 年　12 月　12 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：　　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 (1) {

    /* wait for a connection request */
    childfd = accept(parentfd, (struct sockaddr *) &clientaddr, &clientlen);
    if (childfd < 0)
        error("ERROR on accept");

    pthread_t tid;
    if (pthread_create(&tid, NULL, HandleThread, &childfd) != 0) {
        close(childfd);
        error("Error: client create thread failed\n");
    }
    else {
        printf("Server: Connection succeed\n");
    }
}
```

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。

- 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）

```
if ((stream = fdopen(childfd, "r+")) == NULL) {···

printf("------------------------------\n");
// get the HTTP request line
fgets(buf, BUFSIZE, stream);
printf("%s", buf);
sscanf(buf, "%s %s %s\n", method, url, version);

if (strcasecmp(method, "GET") && strcasecmp(method, "POST")) {···

// read (and ignore) the HTTP headers
int content_length = -1;
fgets(buf, BUFSIZE, stream);
printf("%s", buf);
while(strcmp(buf, "\r\n")) {···
printf("content_length = %d\n", content_length);

// can't find content_length
if (!strcasecmp(method, "POST") && content_length == -1) {   ···

if (!strstr(url, "cgi-bin")) { // static content···
else { // dynamic content···

if (!strcasecmp(method, "GET")) {···
else if (!strcasecmp(method, "POST")) {···
```

第一步，将接受到的 client 的 socket descriptor 以流的形式打开，便于后面字符串处理。

第二步，得到 HTTP 的请求报文的请求行。

第三步，分离请求行的方法，URL 和版本。

第四步，读取请求报文的首部行信息，如果有 content length 字段则读取其值

第五步，判断请求报文的 method 是否为 GET 或者 POST，如果不是，发送错误信息.

第六步，判断是否有 cgi

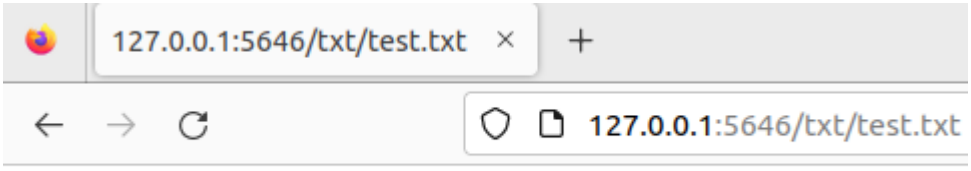
第七步，如果为 GET 方法做出相应响应

第八步，如果为 POST 方法做出相应响应

- 服务器运行后，用 netstat –an 显示服务器的监听端口

```
zarin@ZaricCarpathia:~/Lab8/webserver$ netstat -an | grep 5646
tcp        0      0 127.0.0.1:5646          0.0.0.0:*               LISTEN
unix  3      [ ]         流         已连接     45646
```

- 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：

```
zarin@ZaricCarpathia:~/Lab8/webserver/doc/txt$ ls
test.txt
```

服务器的相关代码片段：

获得文件路径

```
if (!strstr(url, "cgi-bin")) { // static content
    printf("static content\n");
    is_static = 1;
    strcpy(cgiargs, "");
    strcpy(filename, "doc");
    strcat(filename, url);
    if (url[strlen(url)-1] == '/')
        strcat(filename, "index.html");
}
```

判断请求文件类型

```
    if (is_static) {
        if (strstr(filename, ".html"))
            strcpy(filetype, "text/html");
        else if (strstr(filename, ".gif"))
            strcpy(filetype, "image/gif");
        else if (strstr(filename, ".jpg"))
            strcpy(filetype, "image/jpg");
        else
            strcpy(filetype, "text/plain");
```

传送响应报文头部

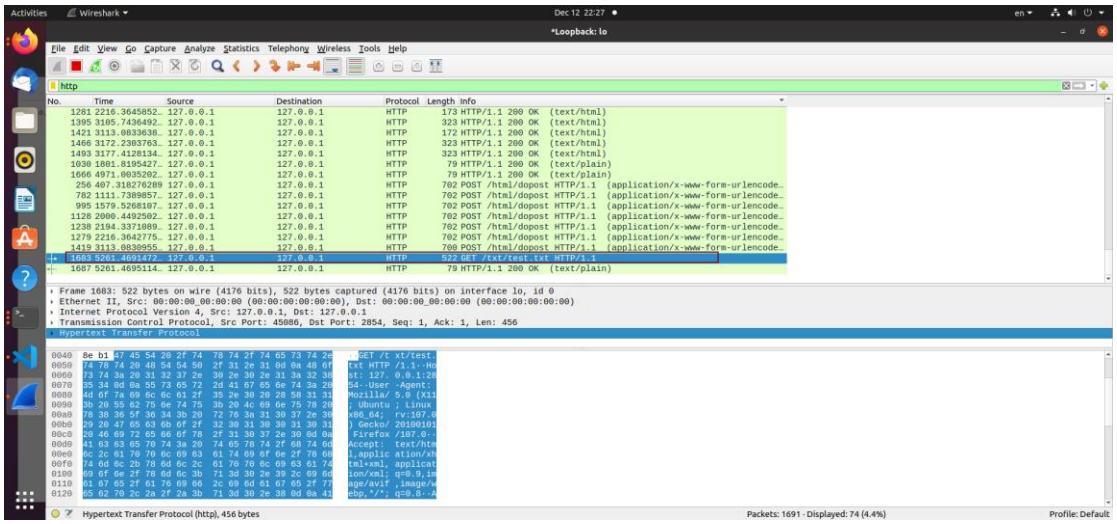
```
            /* print response header */
            fprintf(stream, "HTTP/1.1 200 OK\n");
            fprintf(stream, "Server: Tiny Web Server\n");
            fprintf(stream, "Content-length: %d\n",
(int)sbuf.st_size);
            fprintf(stream, "Content-type: %s\n", filetype);
            fprintf(stream, "\r\n");
            fflush(stream);
```

传送响应报文体部

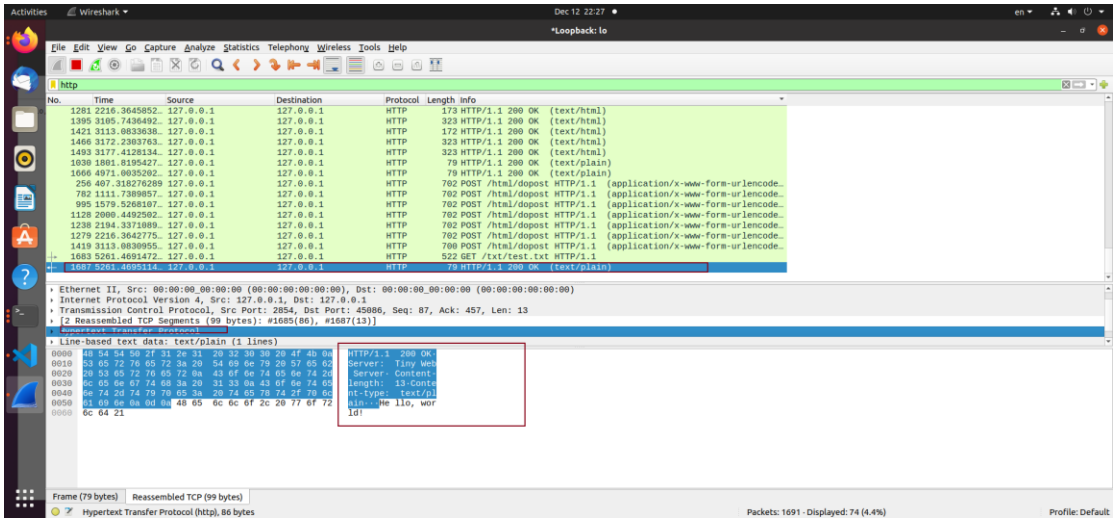
```
            /* Use mmap to return arbitrary-sized response body */
            fd = open(filename, O_RDONLY);
            p = mmap(0, sbuf.st_size, PROT_READ, MAP_PRIVATE, fd,
0);
            fwrite(p, 1, sbuf.st_size, stream);
            munmap(p, sbuf.st_size);
        }
```

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

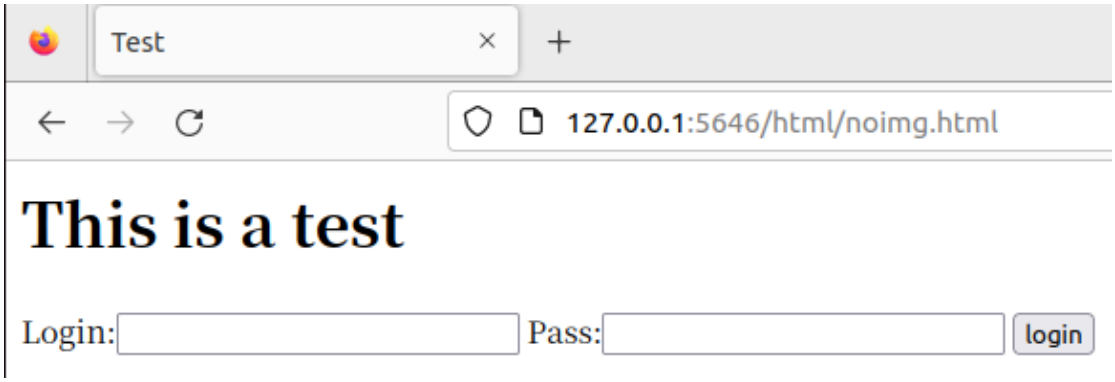
HTTP 请求报文



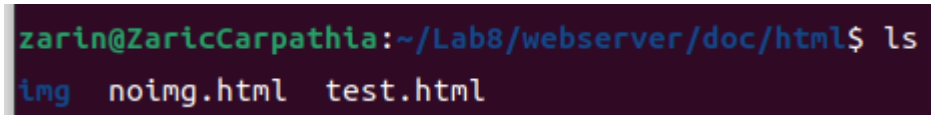
HTTP 响应报文



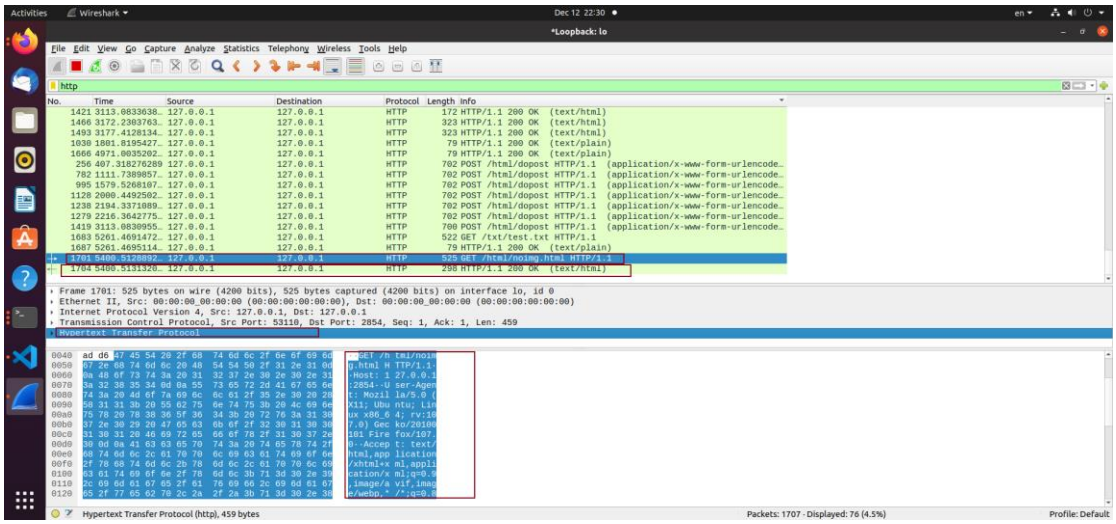
- 浏览器访问只包含文本的 HTML 文件时，浏览器的 URL 地址和显示内容截图。



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）：



- 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。

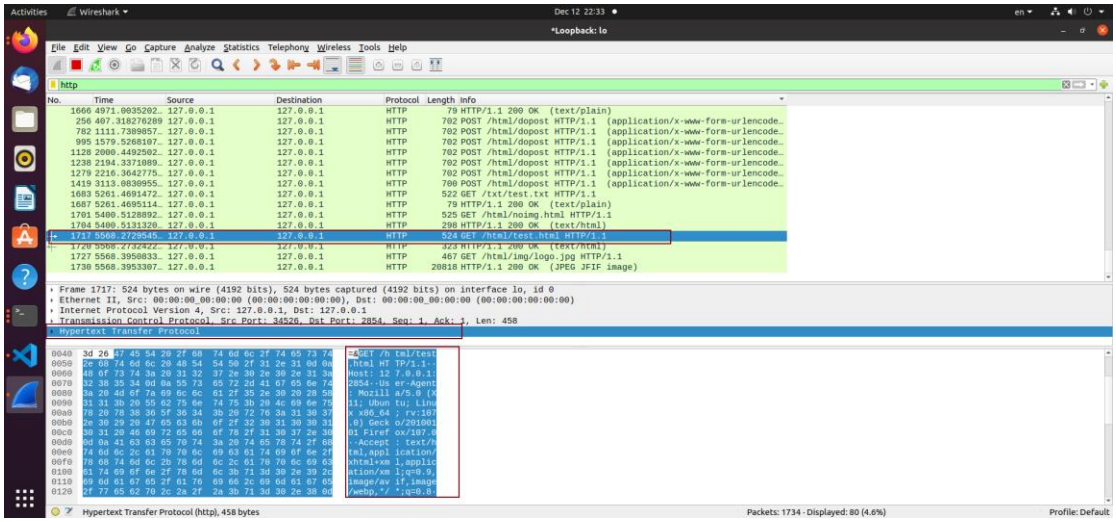
服务器上文件实际存放的路径：

```
zarin@ZaricCarpathia:~/Lab8/webserver/doc/html$ ls
img  noimg.html  test.html
```

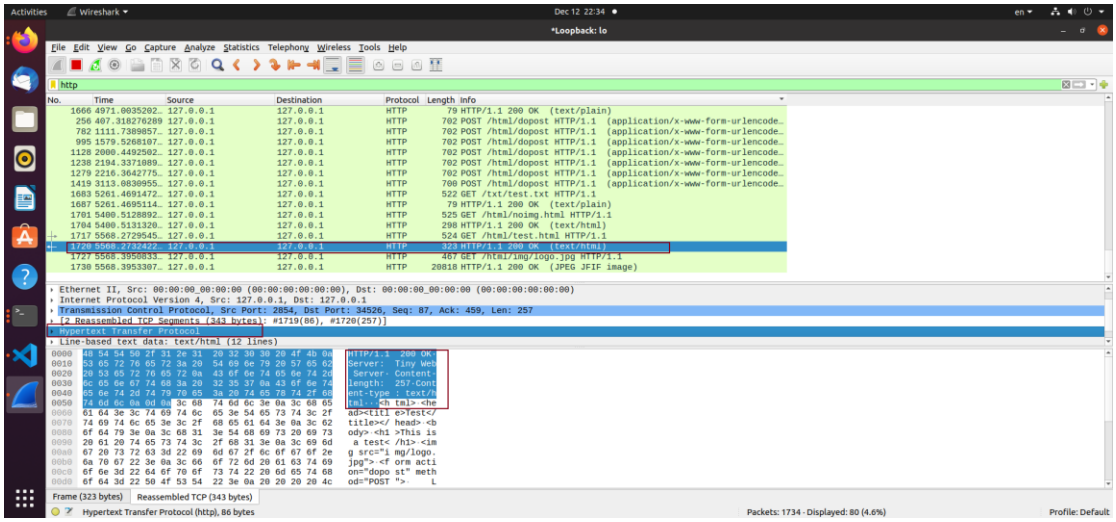
Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML、图片文件的部分内容）：

因为 html 中包含图片，其有两次 http 请求

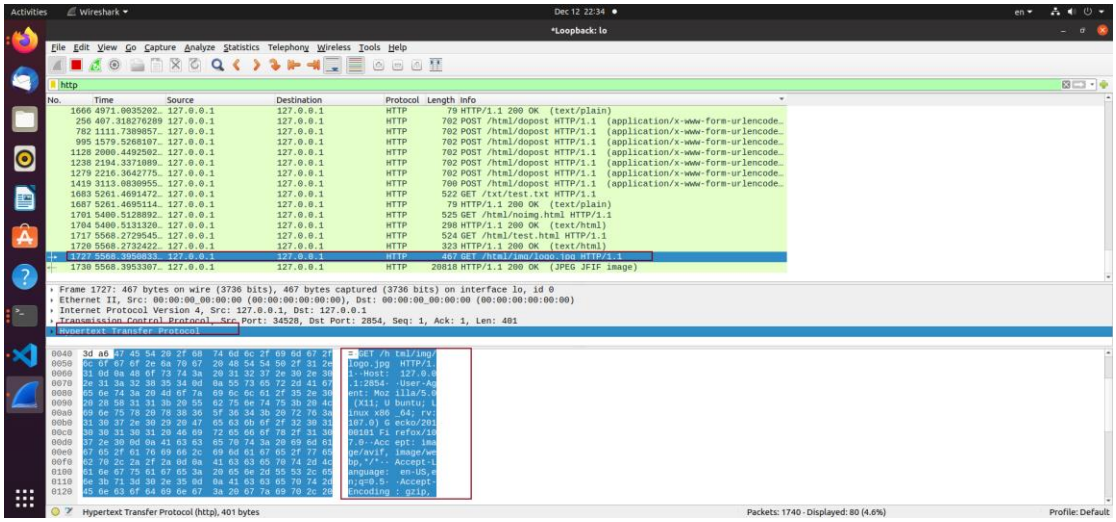
第一次，请求 html 文件



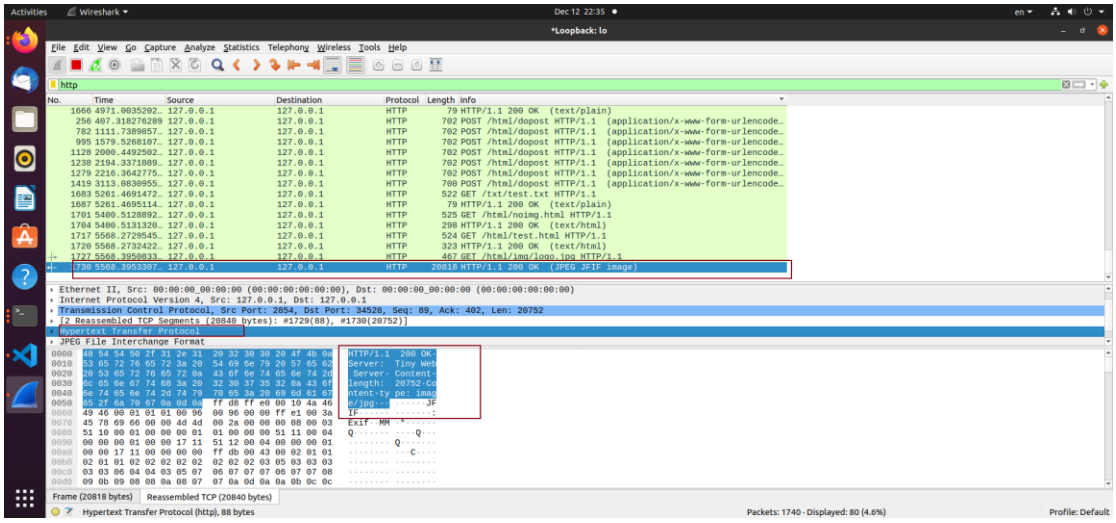
html 响应报文



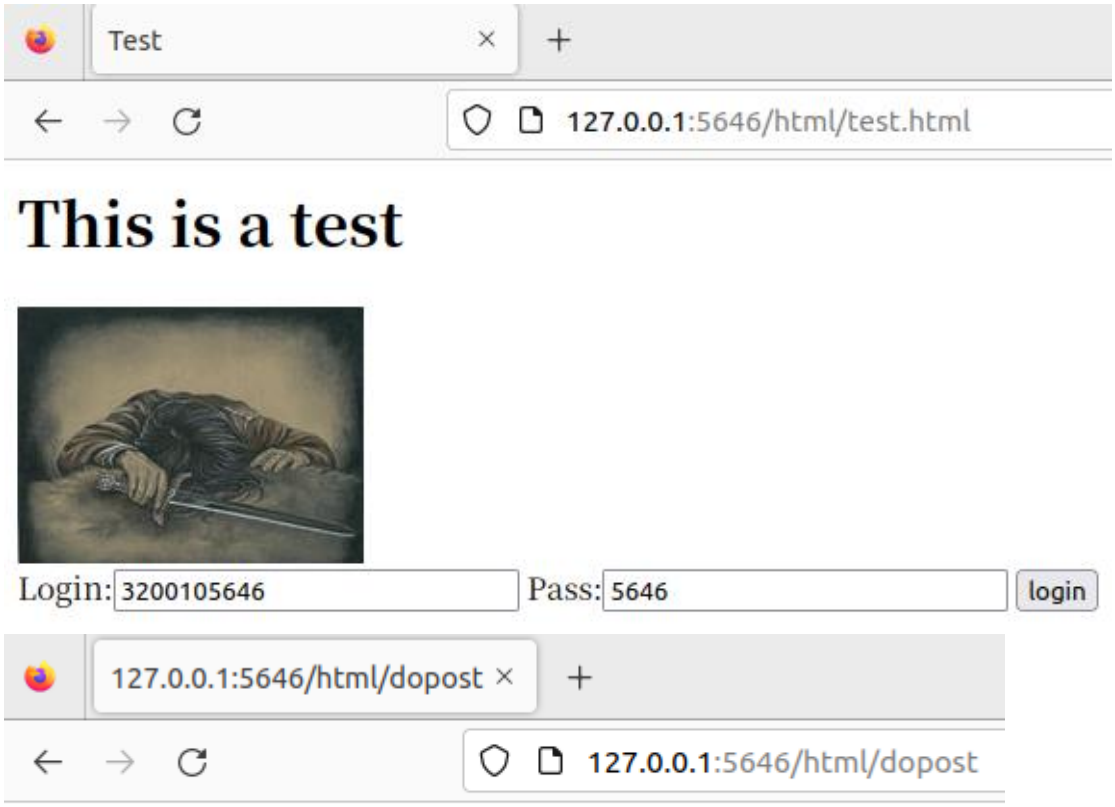
http 图片请求报文



图片响应报文



- 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：

判断是否为 POST 方法

```
 else if (!strcasecmp(method, "POST")) {
        printf("POST!!!!!!!!!!!!!!!!!!!\n");
        printf("filename = %s\n", filename);
```

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，并继续下一步。

```
        if (strcmp(filename, "doc/html/dopost") != 0) {
            // cerror(stream, filename, "404", "Not found",
            //     "Tiny couldn't find this file");
            /* print response header */
            fprintf(stream, "HTTP/1.1 404 Not found\r\n");
            fprintf(stream, "Server: Tiny Web Server\r\n");
            fprintf(stream, "Content-length: %d\r\n",
(int)strlen("<p>404 Not found\r\n"));
            fprintf(stream, "Content-type: text/html\r\n");
            fprintf(stream, "\r\n");
            fprintf(stream, "<p>404 Not found\r\n");
            fflush(stream);

            fclose(stream);
            close(childfd);
```

```
            return;
        }
```

提取 login 和 password

```
        printf("get a line from stream...\n");
        // fprintf(stream, "HTTP/1.1 200 OK\r\n");
        fgets(buf, content_length + 1, stream);
        printf("last buf = %s\n", buf);
        int i;
        int index = -1;
        for (i = 0; i < content_length; i++) {
            if (buf[i] == '&') {
                index = i;
            }
        }
        if (index == -1) {
            printf("error\n");
            exit(0);
        }
        buf[index] = '\0';
        buf[content_length] = '\0';
        char *login;
        char *pass;
        login = buf + 6;
        pass = buf + index + 6;
        printf("login = %s\n", login);
        printf("pass = %s\n", pass);
```

判断登录的用户名和密码是否正确

```
        int msglength = 0;
        char *msg;
        if (strcmp(login, "3200105646") == 0 && strcmp(pass, "5646")
== 0) {
            // login_response(stream, "login succeed");
            // msglength = strlen("login succeed");
            msg = "login succeed";
        }
        else {
            // msglength = strlen("login failed");
            msg = "login failed";
        }
```

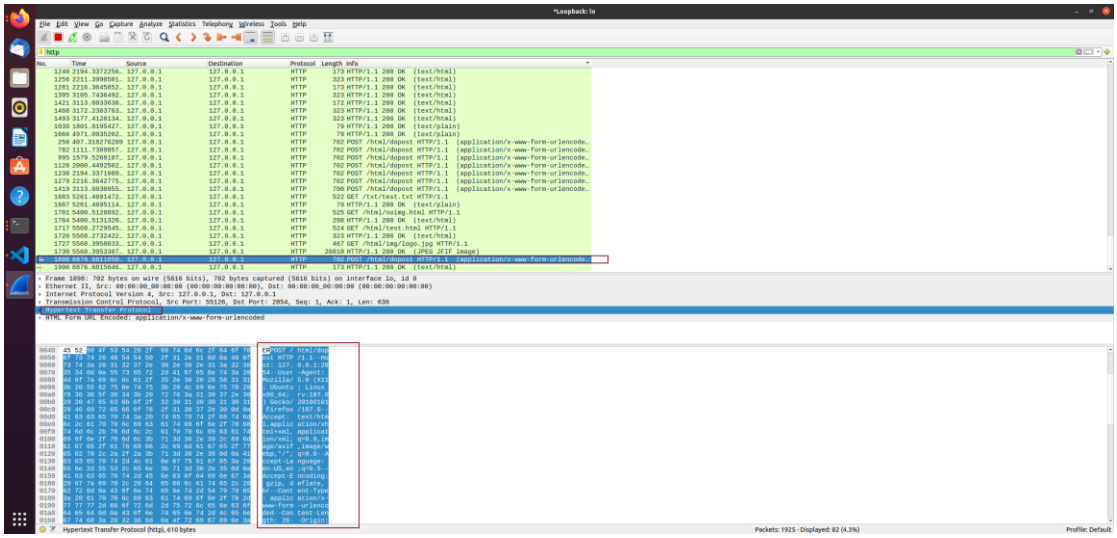
写响应包头部

```
        /* print response header */
        msglength = strlen(msg);
        fprintf(stream, "HTTP/1.1 200 OK\r\n");
        fprintf(stream, "Server: Tiny Web Server\r\n");
```

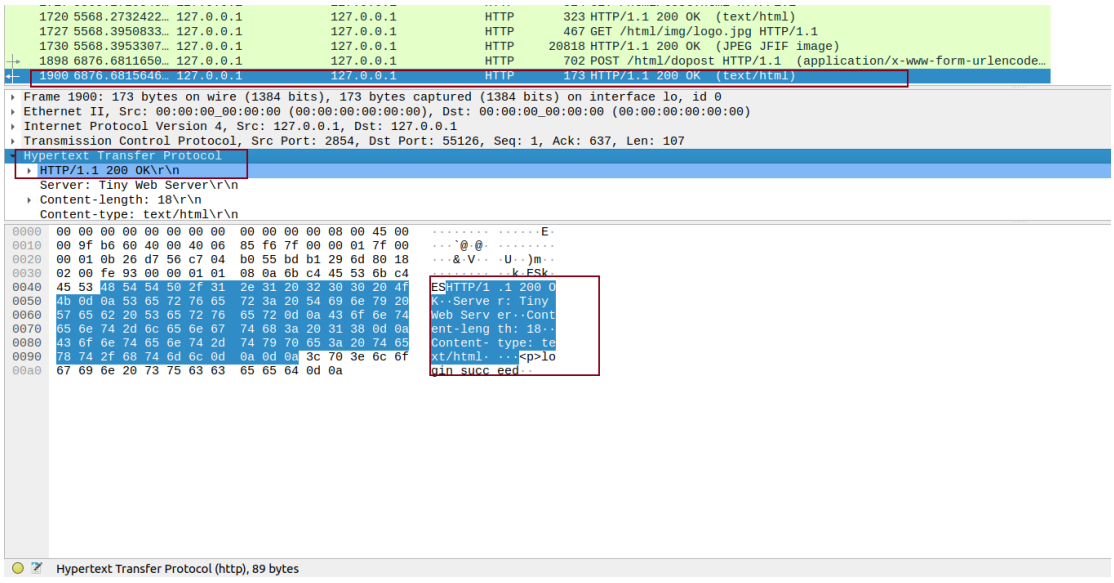
```
        fprintf(stream, "Content-length: %d\r\n",
(int)strlen("<p>\r\n") + msglength);
        fprintf(stream, "Content-type: text/html\r\n");
        fprintf(stream, "\r\n");
        fprintf(stream, "<p>%s\r\n", msg);
        fflush(stream);
    }
```

Wireshark 抓取的数据包截图（HTTP 协议部分）

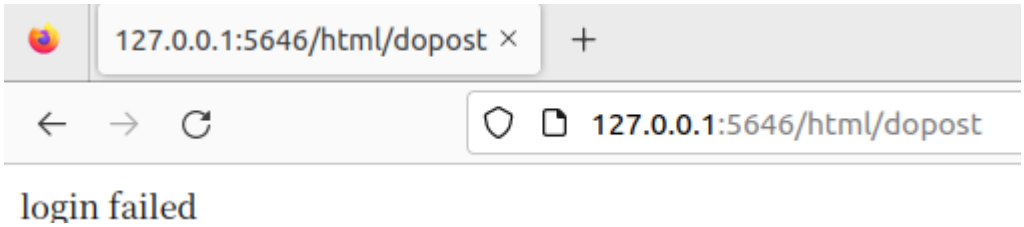
请求包



响应包

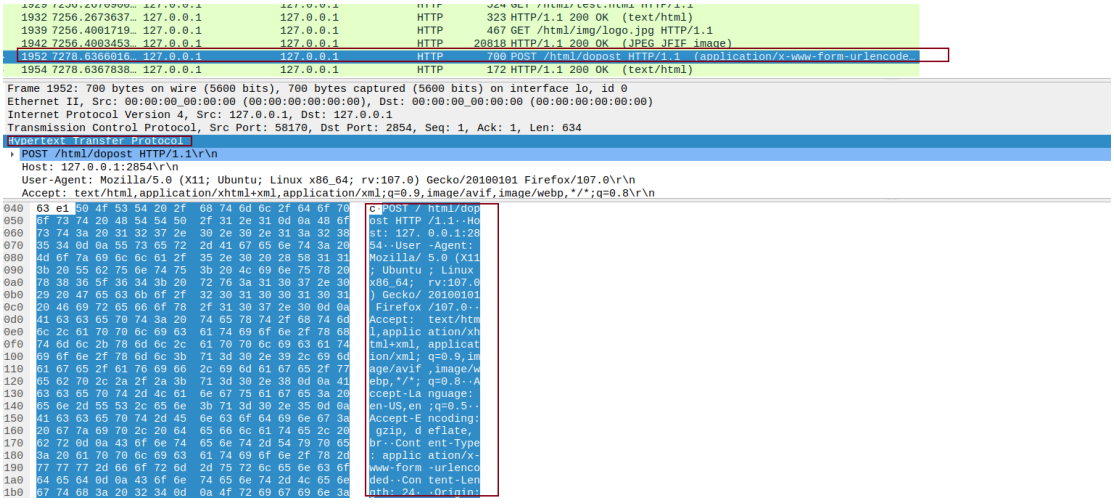


- 浏览器输入错误的登录名或密码，点击登录按钮（login）后的显示截图。

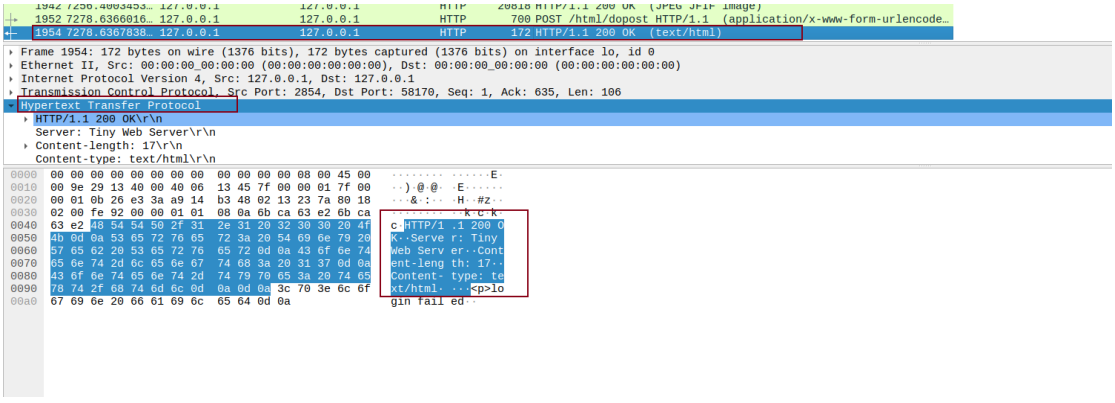


- Wireshark 抓取的数据包截图（HTTP 协议部分）

  请求包



  响应包



- 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）

```
zarin@ZaricCarpathia:~/Lab8/webserver$ netstat -an | grep 5646
tcp        0      0 127.0.0.1:5646          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5646          127.0.0.1:45504         TIME_WAIT
tcp        1      0 127.0.0.1:5646          127.0.0.1:50522         CLOSE_WAIT
tcp        0      0 127.0.0.1:5646          127.0.0.1:45518         TIME_WAIT
tcp      682      0 127.0.0.1:5646          127.0.0.1:55046         CLOSE_WAIT
tcp      504      0 127.0.0.1:5646          127.0.0.1:59834         CLOSE_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？

  通过一个空行，空行中为\r\n，来进行分隔。

- 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？

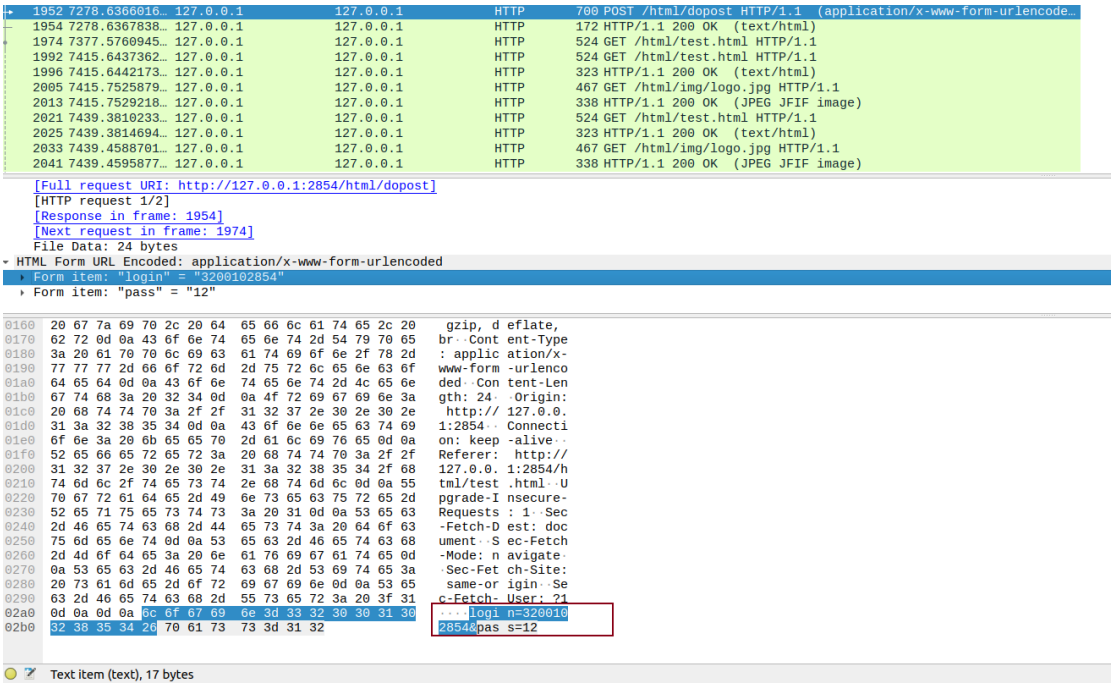
  浏览器是根据 content type 字段来判断文件类型的。

- HTTP 协议的头部是不是一定是文本格式？体部呢？

  HTTP 协议的头部一定是文本格式，体部不一定是文本格式，比如视频和音频以字节流。

- 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？

  POST 方法传递的数据放在体部；



login 字段和 pass 字段是通过&符号连接起来的；

字段属性和字段值是通过=连接起来的。

七、 讨论、心得