实验2：配置Web服务器，分析HTTP交互过程

2212452 孟启轩 计算机科学与技术

实验要求

（1）搭建Web服务器（自由选择系统），并制作简单的Web页面，包含简单文本信息（至少包含专业、学号、姓名）、自己的LOGO、自我介绍的音频信息。 （2）通过浏览器获取自己编写的Web页面，使用Wireshark捕获浏览器与Web服务器的交互过程，使用Wireshark过滤器使其只显示HTTP协议。 （3）提交HTML文档、Wireshark捕获文件和实验报告，对HTTP交互过程进行详细说明。

实验环境

- 操作系统：Windows 11
- 开发工具：Python 3, Flask, edge浏览器
- 网络分析工具：Wireshark

实验内容

1. 搭建Web服务器

使用Python的Flask框架搭建Web服务器，选择HTTP协议作为通信方式。Flask是一个轻量级的Web框架，适用于本次实验的基本需求。

Flask安装

使用以下命令安装Flask：

```
pip install flask
```

编写Flask程序

创建一个名为`me.py`的文件，编写Flask应用程序代码，实现Web服务器的基本配置：

```
from flask import Flask, render_template

app = Flask(__name__)

# 首页路由，渲染index.html模板
@app.route('/')
def home():
    return render_template('index.html')

if __name__ == '__main__':
    # 运行Flask开发服务器
    app.run(host='0.0.0.0', port=5000, debug=True)
```

制作Web页面

创建一个名为`index.html`的页面，展示基本的个人信息，包括学校、专业、学号、姓名等。使用HTML编写页面，并存放在`/templates`目录下。

```
<!DOCTYPE html>
<html lang="zh-CN">

<head>
    <meta charset="UTF-8">
    <title>个人主页</title>
</head>

<body>
    <h1>个人信息</h1>
    <p>学校：南开大学</p>
    <p>专业：计算机科学与技术</p>
    <p>学号：2212452</p>
    <p>姓名：孟启轩</p>
    <h2>个人LOGO</h2>
    <img src="{{ url_for('static', filename='logo.png') }}" alt="个人LOGO"
width="300">
    <h2>自我介绍音频</h2>
    <audio controls>
        <source src="{{ url_for('static', filename='introduction.mp3') }}"
type="audio/mpeg">
        您的浏览器不支持音频标签。
    </audio>
</body>

</html>
```

2. 运行Web服务器

在命令行或终端中运行以下命令启动服务器：

```
python me.py
```

得到：

```
 * Serving Flask app 'me'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
```

```
 * Running on http://10.136.53.64:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 829-248-213
127.0.0.1 - - [24/Oct/2024 00:12:53] "GET /static/introduction.mp3 HTTP/1.1" 206 -
127.0.0.1 - - [24/Oct/2024 00:13:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Oct/2024 00:13:10] "GET /static/logo.png HTTP/1.1" 304 -
```

在Flask启动时输出两个URL，表示服务器可以通过不同的IP地址访问。

- http://127.0.0.1:5000： 这是本地主机（localhost）的地址，表示只能在运行Flask服务器的计算机上访问这个Web页面。访问这个地址时，实际上是通过环回接口（loopback interface）进行通信，用于本地开发和调试。
- http://10.136.53.64:5000： 这是服务器在局域网中的IP地址，可以通过局域网中的其他设备访问这个地址。其他设备可以使用这个IP地址和端口来访问Flask服务器托管的Web页面，前提是这些设备与服务器位于同一个局域网，并且服务器的防火墙没有阻止外部访问。

通过这两个URL，可以选择在本地开发调试时使用127.0.0.1，或者在局域网中测试访问使用服务器的局域网IP地址。

点击链接 http://127.0.0.1:5000 进入网页

3. 使用Wireshark捕获浏览器与Web服务器的交互过程

在Wireshark里选择`Adapter for loopback traffic capture`网络接口进行捕获，它是本机的环回接口。并在过滤器输入`host 127.0.0.1`，只查看127.0.0.1的数据包。

- **客户端和服务器之间的连接建立需要TCP三次握手**

- **TCP三次握手是在HTTP交互前进行的，确保传输的可靠性。**

No.	Time	Source	Destination	Protoco	Lengt	Info
1	0.000000	127.0.0.1	127.0.0.1	TLSv1.2	229	Application Data
2	0.000036	127.0.0.1	127.0.0.1	TCP	44	54533 → 62128 [ACK] Seq=1 Ack=186 Win=8300 Len=0
3	0.000118	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
4	0.000144	127.0.0.1	127.0.0.1	TCP	44	62129 → 62130 [ACK] Seq=1 Ack=157 Win=8324 Len=0
5	0.000341	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
6	0.000363	127.0.0.1	127.0.0.1	TCP	44	62130 → 62129 [ACK] Seq=157 Ack=148 Win=8331 Len=0
7	0.000450	127.0.0.1	127.0.0.1	TLSv1.2	220	Application Data
8	0.000474	127.0.0.1	127.0.0.1	TCP	44	62128 → 54533 [ACK] Seq=186 Ack=177 Win=8304 Len=0
9	4.138583	127.0.0.1	127.0.0.1	TCP	56	56025 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
10	4.138632	127.0.0.1	127.0.0.1	TCP	56	5000 → 56025 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
11	4.138670	127.0.0.1	127.0.0.1	TCP	44	56025 → 5000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
12	4.138869	127.0.0.1	127.0.0.1	TCP	56	56026 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13	4.138898	127.0.0.1	127.0.0.1	TCP	56	5000 → 56026 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
14	4.138923	127.0.0.1	127.0.0.1	TCP	44	56026 → 5000 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
15	4.150318	127.0.0.1	127.0.0.1	HTTP	1345	GET / HTTP/1.1
16	4.150406	127.0.0.1	127.0.0.1	TCP	44	5000 → 56025 [ACK] Seq=1 Ack=1302 Win=2159872 Len=0
17	4.156333	127.0.0.1	127.0.0.1	TCP	218	5000 → 56025 [PSH, ACK] Seq=1 Ack=1302 Win=2159872 Len=174 [TCP PDU reassembled in 19]
18	4.156382	127.0.0.1	127.0.0.1	TCP	44	56025 → 5000 [ACK] Seq=1302 Ack=175 Win=2161152 Len=0
19	4.156420	127.0.0.1	127.0.0.1	HTTP	607	HTTP/1.1 200 OK (text/html)
20	4.156437	127.0.0.1	127.0.0.1	TCP	44	56025 → 5000 [ACK] Seq=1302 Ack=738 Win=2160384 Len=0
21	4.174199	127.0.0.1	127.0.0.1	TCP	44	5000 → 56025 [FIN, ACK] Seq=738 Ack=1302 Win=2159872 Len=0
22	4.174260	127.0.0.1	127.0.0.1	TCP	44	56025 → 5000 [ACK] Seq=1302 Ack=739 Win=2160384 Len=0
23	4.210390	127.0.0.1	127.0.0.1	TCP	56	56027 → 10007 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
24	4.210458	127.0.0.1	127.0.0.1	TCP	56	10007 → 56027 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
25	4.210511	127.0.0.1	127.0.0.1	TCP	44	56027 → 10007 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
26	4.210895	127.0.0.1	127.0.0.1	HTTP	669	GET /download HTTP/1.1

  - **第一次握手**：首先，客户端的TCP程序向服务器的TCP程序发送一个连接请求报文段。这个报文段的SYN标志位被置为1，表示这是一个连接请求（SYN报文段），且不包含数据。客户端随机生成一个初始序号（假设为client_isn）并将其填入序号字段。该报文段经过运输层和网络层的封装，最终作为一个IP数据报发送至服务器。
  - **第二次握手**：服务器的网络层接收到包含SYN的IP数据报后，抽取出TCP报文并交给运输层处理，同时为该TCP连接分配资源（如发送和接收缓存）。服务器确认连接请求，并生成一个带有SYN和ACK标志位的响应报文段（SYN ACK报文段）。服务器也随机选择一个初始序号（假设为server_isn）填入序号字段，同时在确认字段中填入client_isn + 1，以表示已确认客户端的初始序号。该报文段传达的意思是："我收到了你的连接请求并允许连接，我的初始序号是server_isn"。
  - **第三次握手**：客户端接收到服务器的SYN ACK报文后，也为连接分配资源，并发送一个确认报文段给服务器。这个报文段的SYN标志位为0（表明连接已建立），ACK标志位为1，确认字段设置为server_isn + 1（表示成功接收服务器的初始序号）。报文的序号字段被设置为client_isn + 1。同时，这一报文段可以携带数据，例如HTTP请求可能就包含在此报文段中，随即发送至服务器。
  - 握手过程中传送的包里不包含数据，可以看到，`Len=0`。
  - 成功完成三次握手后，客户端和服务器进入数据传输阶段，建立的TCP连接处于ESTABLISHED状态。
  - Flask默认使用HTTP/1.1协议来处理请求和响应。

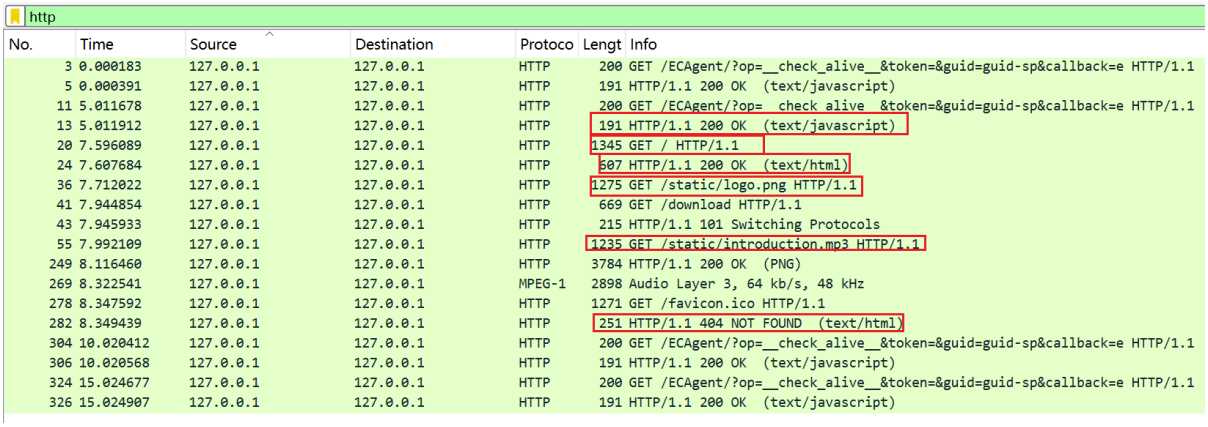
- **PSH表示有数据传输**

56	7.992245	127.0.0.1	127.0.0.1	TCP	44	5000 → 64934 [ACK] Seq=1 Ack=1192 Win=2160128 Len=0
57	8.109731	127.0.0.1	127.0.0.1	TCP	407	5000 → 64933 [PSH, ACK] Seq=1 Ack=1232 Win=2159872 Len=363 [TCP PDU reassembled in 249]
58	8.109777	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=364 Win=326912 Len=0
59	8.109825	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=364 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
60	8.109848	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=8556 Win=318720 Len=0
61	8.109892	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=8556 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
62	8.109919	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=16748 Win=310528 Len=0
63	8.109953	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=16748 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
64	8.109971	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=24940 Win=302336 Len=0
65	8.109996	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=24940 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
66	8.110013	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=33132 Win=294144 Len=0
67	8.110042	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=33132 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
68	8.110076	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=41324 Win=285952 Len=0
69	8.110113	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=41324 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
70	8.110133	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=49516 Win=277760 Len=0
71	8.110165	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=49516 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
72	8.110183	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=57708 Win=269568 Len=0
73	8.110208	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=57708 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
74	8.110225	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=65900 Win=327424 Len=0
75	8.110251	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=65900 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]
76	8.110286	127.0.0.1	127.0.0.1	TCP	44	64933 → 5000 [ACK] Seq=1232 Ack=74092 Win=319232 Len=0
77	8.110335	127.0.0.1	127.0.0.1	TCP	8236	5000 → 64933 [PSH, ACK] Seq=74092 Ack=1232 Win=2159872 Len=8192 [TCP PDU reassembled in 249]

- **HTTP请求和响应**

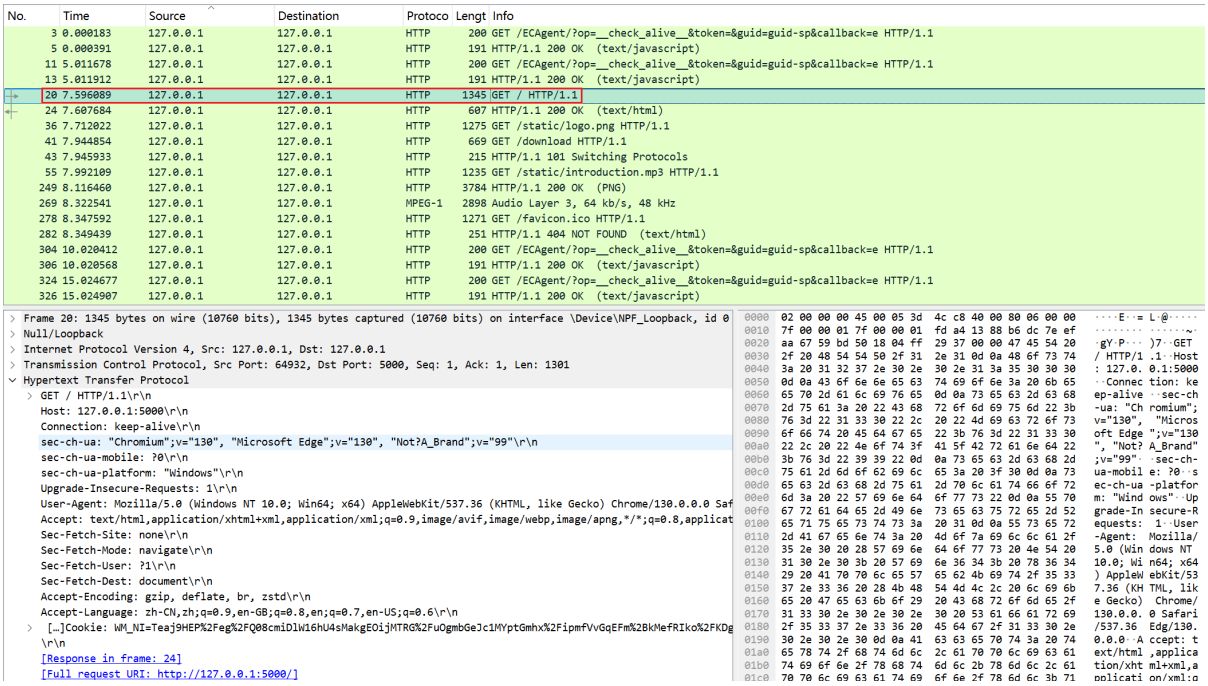
  - **HTTP的请求和响应在TCP连接建立后进行，通过TCP传输应用层数**据。

- 在过滤器中输入http



No.	Time	Source	Destination	Protoco	Lengt	Info
3	0.000183	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
5	0.000391	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
11	5.011678	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
13	5.011912	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
20	7.596089	127.0.0.1	127.0.0.1	HTTP	1345	GET / HTTP/1.1
24	7.607684	127.0.0.1	127.0.0.1	HTTP	607	HTTP/1.1 200 OK (text/html)
36	7.712022	127.0.0.1	127.0.0.1	HTTP	1275	GET /static/logo.png HTTP/1.1
41	7.944854	127.0.0.1	127.0.0.1	HTTP	669	GET /download HTTP/1.1
43	7.945933	127.0.0.1	127.0.0.1	HTTP	215	HTTP/1.1 101 Switching Protocols
55	7.992109	127.0.0.1	127.0.0.1	HTTP	1235	GET /static/introduction.mp3 HTTP/1.1
249	8.116460	127.0.0.1	127.0.0.1	HTTP	3784	HTTP/1.1 200 OK (PNG)
269	8.322541	127.0.0.1	127.0.0.1	MPEG-1	2898	Audio Layer 3, 64 kb/s, 48 kHz
278	8.347592	127.0.0.1	127.0.0.1	HTTP	1271	GET /favicon.ico HTTP/1.1
282	8.349439	127.0.0.1	127.0.0.1	HTTP	251	HTTP/1.1 404 NOT FOUND (text/html)
304	10.020412	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
306	10.020568	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
324	15.024677	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
326	15.024907	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)

- **HTTP请求报文包含请求行、请求头、空行和可选的请求体。**
  - **请求行**：包括HTTP方法（GET、POST、PUT、DELETE等）、请求的资源路径、HTTP版本（通常为HTTP/1.1或HTTP/2.0）。
  - **请求头**：包含多个键值对的请求头字段，用于传递请求的附加信息。
    - 常见的请求头：Host、User-Agent、Accept、Connection。
  - **请求体**：用于传递请求数据，通常用于POST或PUT请求。



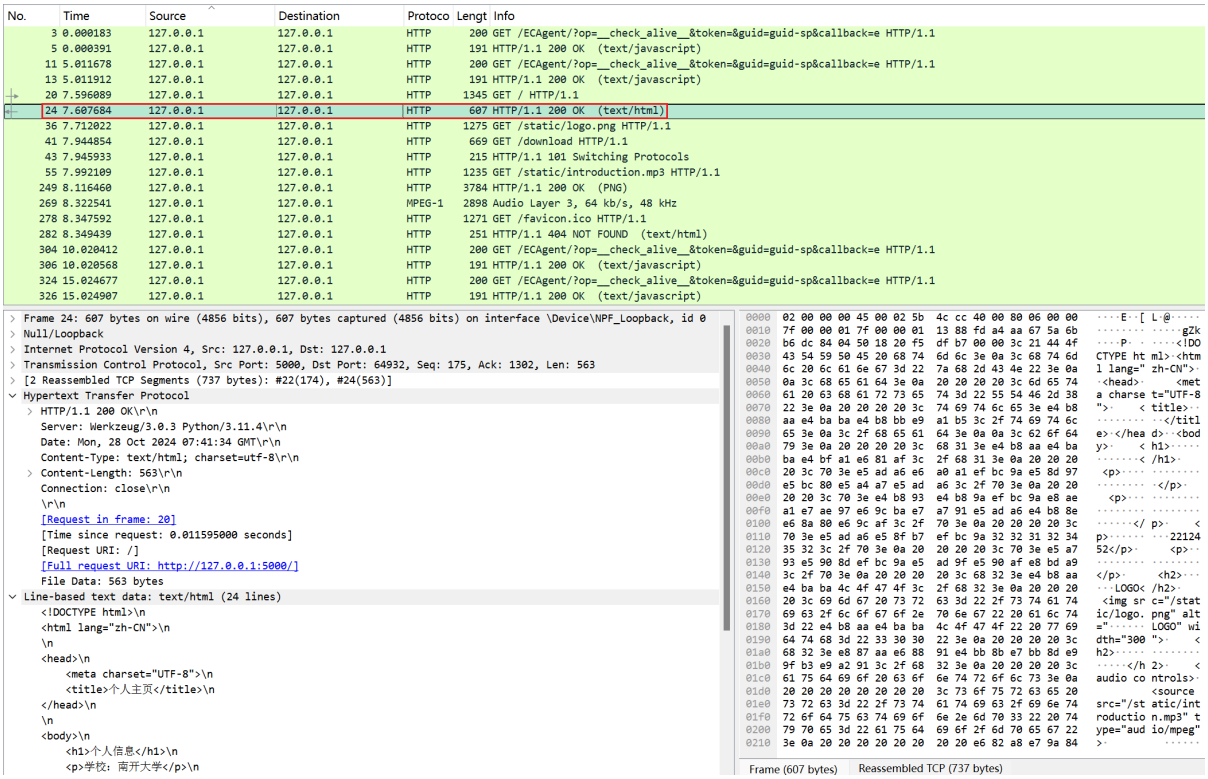
No.	Time	Source	Destination	Protoco	Lengt	Info
3	0.000183	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
5	0.000391	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
11	5.011678	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
13	5.011912	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
20	7.596089	127.0.0.1	127.0.0.1	HTTP	1345	GET / HTTP/1.1
24	7.607684	127.0.0.1	127.0.0.1	HTTP	607	HTTP/1.1 200 OK (text/html)
36	7.712022	127.0.0.1	127.0.0.1	HTTP	1275	GET /static/logo.png HTTP/1.1
41	7.944854	127.0.0.1	127.0.0.1	HTTP	669	GET /download HTTP/1.1
43	7.945933	127.0.0.1	127.0.0.1	HTTP	215	HTTP/1.1 101 Switching Protocols
55	7.992109	127.0.0.1	127.0.0.1	HTTP	1235	GET /static/introduction.mp3 HTTP/1.1
249	8.116460	127.0.0.1	127.0.0.1	HTTP	3784	HTTP/1.1 200 OK (PNG)
269	8.322541	127.0.0.1	127.0.0.1	MPEG-1	2898	Audio Layer 3, 64 kb/s, 48 kHz
278	8.347592	127.0.0.1	127.0.0.1	HTTP	1271	GET /favicon.ico HTTP/1.1
282	8.349439	127.0.0.1	127.0.0.1	HTTP	251	HTTP/1.1 404 NOT FOUND (text/html)
304	10.020412	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
306	10.020568	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
324	15.024677	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
326	15.024907	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)

```
Frame 20: 1345 bytes on wire (10760 bits), 1345 bytes captured (10760 bits) on interface \Device\NPF_Loopback, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 64932, Dst Port: 5000, Seq: 1, Ack: 1, Len: 1301
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: 127.0.0.1:5000\r\n
  Connection: keep-alive\r\n
  sec-ch-ua: "Chromium";v="130", "Microsoft Edge";v="130", "Not?A_Brand";v="99"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Windows"\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Saf
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicat
  Sec-Fetch-Site: none\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Accept-Encoding: gzip, deflate, br, zstd\r\n
  Accept-Language: zh-CN,zh;q=0.9,en-GB;q=0.8,en;q=0.7,en-US;q=0.6\r\n
  […]Cookie: WM_NI=Teaj9HEP%2Feg%2FQ08cmiDlW16hU4sMakgEOijMTRG%2FuOgmbGeJc1MYptGmhx%2FipmfVvGqEFm%2BkMefRIko%2FKDg
  \r\n
  [Response in frame: 24]
  [Full request URI: http://127.0.0.1:5000/]
```

```
0000  02 00 00 00 45 00 05 3d  4c c8 40 00 80 06 00 00   ····E··= L·@·····
0010  7f 00 00 01 7f 00 00 01  fd a4 13 88 b6 dc 7e ef   ········ ·····~·
0020  aa 67 59 bd 50 18 04 ff  29 37 00 00 47 45 54 20   ·gY·P··· )7··GET
0030  2f 20 48 54 54 50 2f 31  2e 31 0d 0a 48 6f 73 74   / HTTP/1 .1··Host
0040  3a 20 31 32 37 2e 30 2e  30 2e 31 3a 35 30 30 30   : 127.0. 0.1:5000
0050  0d 0a 43 6f 6e 6e 65 63  74 69 6f 6e 3a 20 6b 65   ··Connec tion: ke
0060  65 70 2d 61 6c 69 76 65  0d 0a 73 65 63 2d 63 68   ep-alive ··sec-ch
0070  2d 75 61 3a 20 22 43 68  72 6f 6d 69 75 6d 22 3b   -ua: "Ch romium";
0080  76 3d 22 31 33 30 22 2c  20 22 4d 69 63 72 6f 73   v="130",  "Micros
0090  6f 66 74 20 45 64 67 65  22 3b 76 3d 22 31 33 30   oft Edge ";v="130
00a0  22 2c 20 22 4e 6f 74 3f  41 5f 42 72 61 6e 64 22   ", "Not? A_Brand"
00b0  3b 76 3d 22 39 39 22 0d  0a 73 65 63 2d 63 68 2d   ;v="99"· ·sec-ch-
00c0  75 61 2d 6d 6f 62 69 6c  65 3a 20 3f 30 0d 0a 73   ua-mobil e: ?0··s
00d0  65 63 2d 63 68 2d 75 61  2d 70 6c 61 74 66 6f 72   ec-ch-ua -platfor
00e0  6d 3a 20 22 57 69 6e 64  6f 77 73 22 0d 0a 55 70   m: "Wind ows"··Up
00f0  67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52   grade-In secure-R
0100  65 71 75 65 73 74 73 3a  20 31 0d 0a 55 73 65 72   equests:  1··User
0110  2d 41 67 65 6e 74 3a 20  4d 6f 7a 69 6c 6c 61 2f   -Agent:  Mozilla/
0120  35 2e 30 20 28 57 69 6e  64 6f 77 73 20 4e 54 20   5.0 (Win dows NT
0130  31 30 2e 30 3b 20 57 69  6e 36 34 3b 20 78 36 34   10.0; Wi n64; x64
0140  29 20 41 70 70 6c 65 57  65 62 4b 69 74 2f 35 33   ) AppleW ebKit/53
0150  37 2e 33 36 20 28 4b 48  54 4d 4c 2c 20 6c 69 6b   7.36 (KH TML, lik
0160  65 20 47 65 63 6b 6f 29  20 43 68 72 6f 6d 65 2f   e Gecko)  Chrome/
0170  31 33 30 2e 30 2e 30 2e  30 20 53 61 66 61 72 69   130.0.0. 0 Safari
0180  2f 35 33 37 2e 33 36 20  45 64 67 2f 31 33 30 2e   /537.36  Edg/130.
0190  30 2e 30 2e 30 0d 0a 41  63 63 65 70 74 3a 20 74   0.0.0··A ccept: t
01a0  65 78 74 2f 68 74 6d 6c  2c 61 70 70 6c 69 63 61   ext/html ,applica
01b0  74 69 6f 6e 2f 78 68 74  6d 6c 2b 78 6d 6c 2c 61   tion/xht ml+xml,a
01c0  70 70 6c 69 63 61 74 69  6f 6e 2f 78 6d 6c 3b 71   pplicati on/xml;q
```

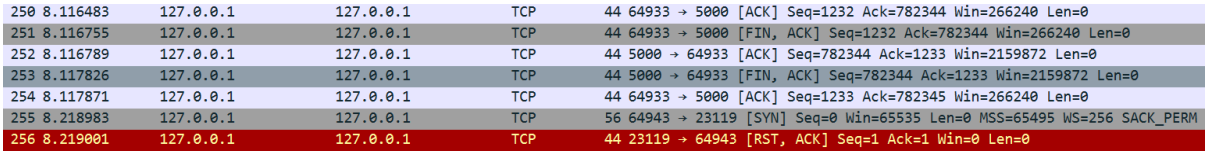
- **HTTP响应报文包含状态行、响应头、空行和可选的响应体。**
  - **状态行**：包括HTTP版本、状态码、状态描述。
    - 状态码：表示请求的处理结果，常见的有： 200 OK：请求成功。 404 Not Found：请求的资源未找到。 500 Internal Server Error：服务器内部错误。
  - **响应头**：包含多个键值对的响应头字段，用于传递响应的附加信息。
    - 常见的响应头： Content-Type：响应内容的媒体类型（如text/html）。 Content-Length：响应体的长度（字节数）。 Connection：连接的状态（如close或keep-alive）。

- **响应体**：用于传递实际的响应数据，如HTML内容或JSON数据。



- **客户端和服务器之间关闭TCP连接需要四次挥手**

  - **TCP四次挥手在HTTP通信完成后进行，用于关闭传输层连接，释放资源。**



  - **第一次挥手**：客户端向服务器发送一个FIN（终止）报文段，请求关闭连接。此时客户端进入FIN_WAIT_1状态。
  - **第二次挥手**：服务器收到FIN报文段后，发送一个ACK报文段确认，并进入CLOSE_WAIT状态，而客户端进入FIN_WAIT_2状态。
  - **第三次挥手**：服务器在完成数据传输后，也发送一个FIN报文段，表示自己准备关闭连接，进入LAST_ACK状态。
  - **第四次挥手**：客户端收到服务器的FIN报文后，发送ACK确认，并进入TIME_WAIT状态，等待一段时间以确保服务器收到ACK后关闭连接。之后客户端关闭连接。
  - 四次挥手完成后，双方的TCP连接完全关闭。

- **HTTP交互过程**

  - 建立TCP连接
  - Web浏览器向Web服务器发送请求行
  - Web浏览器向服务器发送请求头
  - Web服务器应答
  - Web服务器发送应答头
  - Web服务器发送数据
  - Web服务器关闭TCP连接

4. 知识点补充

- **HTTP和HTTPS的区别** HTTP（超文本传输协议）与HTTPS（安全超文本传输协议）之间的主要区别在于安全性和加密方式。HTTP数据在传输过程中未加密，易受到中间人攻击和数据窃取，而HTTPS通过SSL/TLS协议加密数据，确保传输的安全性和完整性。此外，HTTP默认使用端口80，HTTPS则使用端口443。HTTPS在URL中以`https://`开头，并需要获得受信任的SSL/TLS证书，而HTTP则不需要。尽管HTTPS可能会在性能上稍有损失，但其安全优势使其在搜索引擎排名中更受青睐，因此越来越多的网站正在转向HTTPS以提高安全性和用户信任度。
- **HTTP/1.1的特点**
  - **持久连接**：默认情况下，HTTP/1.1会保持TCP连接，允许复用同一个连接进行多个请求和响应。客户端和服务器可以通过Connection: close头部关闭连接。
  - **分块传输编码**：支持分块传输大数据，使服务器可以在响应完成前发送部分数据给客户端。
  - **请求管道化**：可以在发送前将多个请求排队，而无需等待每个请求响应。不过，现代浏览器很少使用这一功能。
  - **更丰富的缓存控制**：增加了如Cache-Control等缓存控制头，支持精确控制资源的缓存策略，改善了浏览器缓存的效率和灵活性。
- **TCP的重要字段**
  - 源端口（Source Port）和目标端口（Destination Port）
    - 用于标识发送方和接收方的应用程序。 端口号的范围为0到65535，其中0到1023为知名端口。
  - 序列号（Sequence Number）：指示该报文段中第一个字节的数据序号。
    - 在数据传输中用于保证数据的有序性和完整性。
  - 确认号（Acknowledgment Number）：用于确认接收到的数据。
    - 表示期望接收的下一个字节的序号，是对接收方已经成功接收到的数据的确认。
  - 数据偏移（Data Offset）：表示TCP报文段头部的长度，以4字节为单位。
    - 用于指示数据部分在报文中的起始位置。
  - 控制标志（Flags）
    - 包含多个标志位，用于控制连接的状态和数据的传输： URG（紧急标志）：表示紧急数据。 ACK（确认标志）：表明确认号字段有效。 PSH（推送标志）：提示接收方尽快将数据传递给应用层。 RST（复位标志）：用于重置连接。 SYN（同步标志）：用于建立连接。 FIN（终止标志）：用于关闭连接。
- **为什么是三次握手而不是两次？** 首先我们要明确，两次握手是必要的。第一次握手，客户端将SYN报文发送到服务器，服务器接收到报文后，即可确认客户端到服务器是可达的；而服务器向客户端发送响应的SYN ACK报文，客户端接收到后，即可确认服务器到客户端也是可达的。至此，连接已经算是建立，那为什么还要有第三次握手呢？
  - 三次握手的必要性不仅在于确认双方的可达性，还涉及同步初始序号（SYN）。在第一次握手中，客户端发送SYN报文告知服务器其初始序号；第二次握手时，服务器返回SYN ACK报文，确认收到客户端的序号并附带服务器的初始序号。第三次握手则确保客户端收到服务器的SYN ACK，完成序号同步。如果省略第三次握手，服务器无法确认客户端已收到其初始序号，可能导致数据传输问题。
  - 三次握手还避免了旧的SYN报文引起资源浪费的情况。客户端重复发送SYN并迅速断开连接时，若旧报文延迟到达，服务器可能会错误地建立新连接。第三次握手可防止这种情况的发生。
- **客户端为什么要等待一段时间再释放资源?**
  - **确保确认报文传递成功**：客户端在发送确认报文（第四次挥手）后，不确定服务器是否成功接收。如果报文丢失，服务器会重传FIN报文，客户端需要再次确认。如果立即释放资源，将无法处理重传的FIN报文。
  - **避免旧数据干扰新连接**：断开连接后，网络中可能仍有未处理的旧数据。如果立即使用相同的四元组（源IP、源端口、目的IP、目的端口）建立新连接，旧数据可能被误认为新连接的数据。因

此，需等待旧数据完全消失后再释放资源，防止干扰。

- **断开连接为什么需要四次挥手？** 断开TCP连接需要四次挥手，因为TCP是全双工的，即双方可以同时发送和接收数据。当一方请求断开连接时（发送FIN），对方会确认（发送ACK），表示接收到断开请求，但这并不意味着对方也没有数据要发送。因此，另一方需要在确认自己的数据发送完毕后，再发送FIN请求断开连接，最后由最初发起断开的一方发送ACK确认，连接才会真正关闭。这确保双方的数据传输完全结束，连接才会安全地释放资源。

总结

在实验过程中，我遇到了一些问题，包括初次配置Flask服务器时未能正确设置静态文件路径，导致页面无法加载LOGO和音频文件。此外，在使用Wireshark捕获HTTP数据包时，由于未正确过滤协议，以及未关闭其他应用进程，导致抓取的数据包中信息杂乱，难以分析。好在问题都一一解决了。

在本次实验中，我成功搭建了基于Flask的Web服务器，并制作了包含个人信息的简单Web页面，深入理解了HTTP协议在浏览器与服务器之间的交互过程。通过使用Wireshark工具捕获HTTP数据包，我观察到客户端与服务器之间的请求和响应流程，增强了对网络通信原理的认识。同时，实验还让我掌握了TCP三次握手和四次挥手的机制，理解了它们在建立和关闭连接过程中的重要性，为后续的网络编程打下了坚实的基础。

部分内容参考(https://www.cnblogs.com/tuyang1129/p/12435772.html "计算机网络——TCP的三次握手与四次挥手（超详细）")