

ExLab1：基于 C 语言实现迷你 http/1.1 服务器

Feng Mingzhe <fmz@mail.bnu.edu.cn>

一、实验目标

在 Windows 下，基于 C 语言，实现一个简单的 http/1.1 服务器，至少包含 200 OK 和 404 Not Found 两个响应。

二、前置知识与技能

2.1 Windows

在开始之前，我们需要达成一个共识，本次实验需要大家在具有 Windows 10 或 Windows 11 的系统中完成，假如你没有 Windows 操作系统的计算机，可以使用 VMware 或者 Crossover 创建一个虚拟机。如果你没有这些条件，请使用图书馆的公用计算机，或联系助教以取得帮助。

2.2 C 语言、IDE 和 MinGW 编译器

本文假设你已经学习过 C 语言的基础知识，能够独立编写 C 程序，并对编译器、命令行有所了解。本文的实验目标是基于 C 语言去实现 http/1.1 协议，因此可能会用到一些原生的 Socket 库，但是没有关系，任何用到的库本文里都会加以说明，甚至会提供示例代码。

关于开发的 IDE，我推荐采用 Microsoft Visual Studio Code（蓝色的 VSCode）来进行开发，这里就不多作介绍了，如果你还没有学过怎么在 VSCode 里配 C 语言的开发环境，可以自己去找个教程。

本文中的代码推荐在 MinGW 编译器中运行，其他的编译器理论上也是可行的，但受限于精力原因无暇测试，因此不对其他的编译器负责。

MinGW64 下载地址：<https://pan.bnu.edu.cn/l/BluZoQ>。

安装方法简单介绍一下，首先把这些文件解压到一个目录（尽量不要选择含有空格的路径），我这里选择了 D:\env\mingw64，如图 1 所示。



图 1

然后像图 2 一样，配置一下环境变量，把 `D:\env\mingw64\bin` 添加进 `PATH`。这样打开命令行，输入 `gcc -v`，有图 3 所示的输出就算成功了。

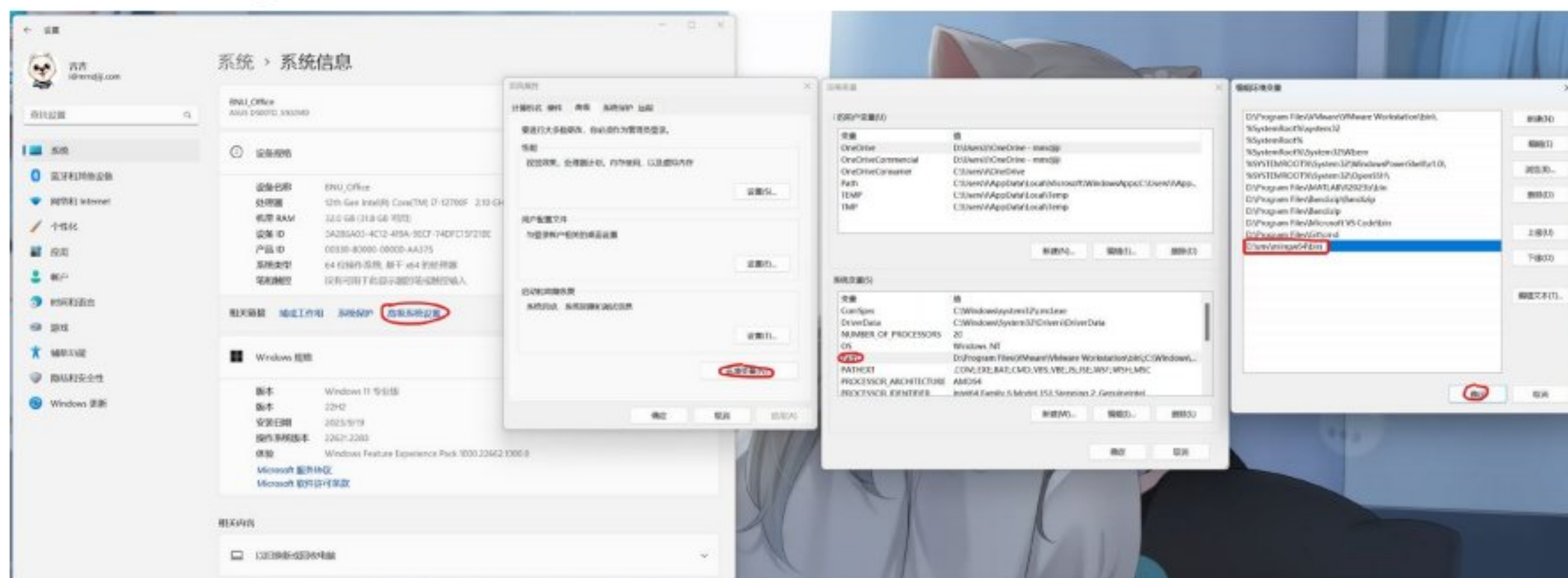


图 2

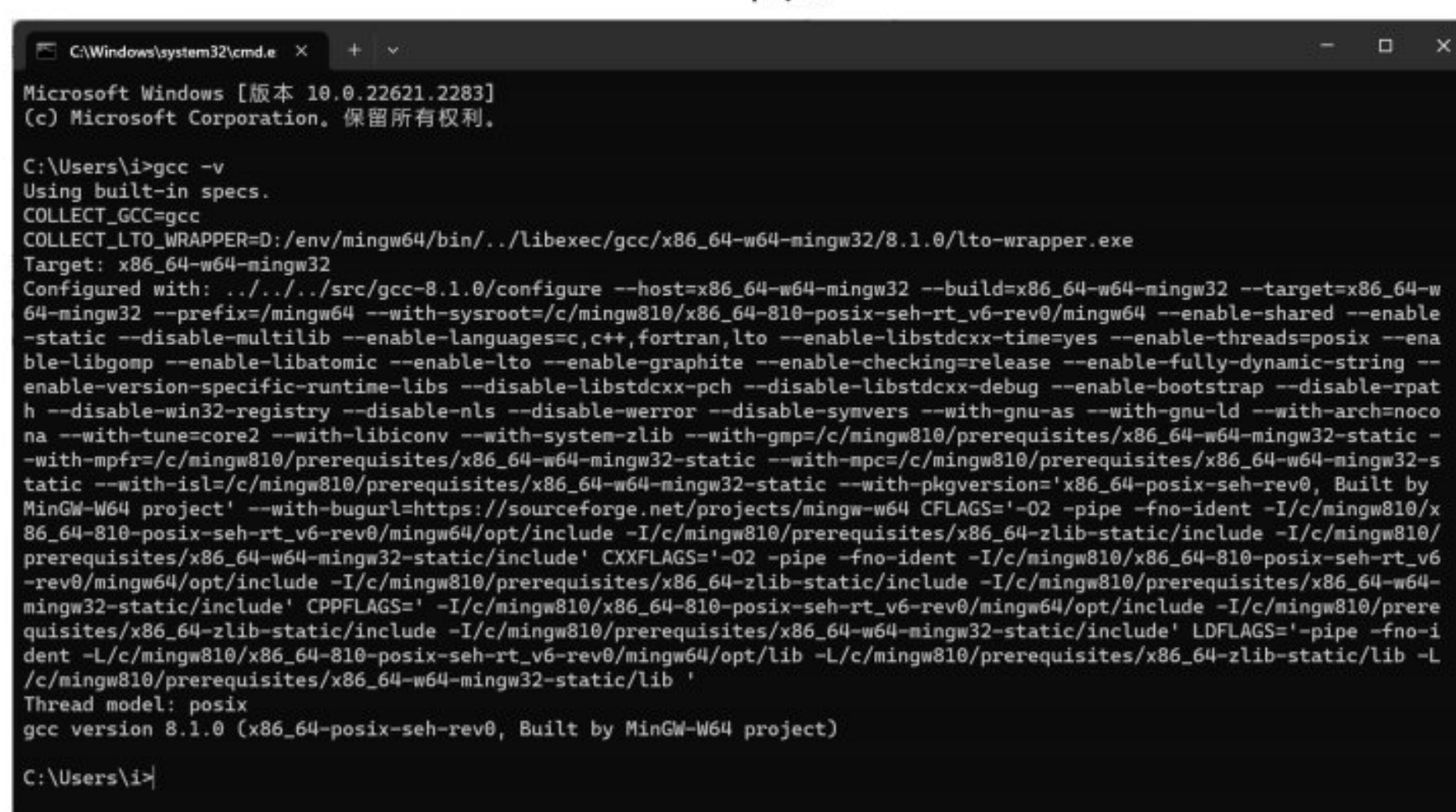


图 3

2.3 Git

Git 是一个代码管理工具，事实上它的功能很多而且比较复杂，但本文实验中只会用到其中的一个指令，那就是：

```
git clone <repo>
```

这行指令的意思是从<repo>拉取代码，<repo>实际上是一个仓库的地址，例如：

```
git clone https://github.com/bnunet/exlab1
```

Git 可以在 <https://git-scm.com/downloads> 中下载，我也建立了师大云盘的镜像，如果上述链接无法访问，可以移步 <https://pan.bnu.edu.cn/l/31XCew>。

2.4 RFC2616

RFC2616 是 http/1.1 协议的定义，也就是任何 http/1.1 的协议都得根据这个标准来实现

为什么他们只提供标准而不是直接实现好代码？因为他们考虑到了使用任何语言来实现这个协议，如果一开始就固定了只有什么语言才能实现某个标准，那就毫无疑问降低了自由度。这也和网络分层的好处一样，任何一层的实现被修改，只要它的接口不变，就不会影响到其他层。

关于 RFC2616 可以参考这个文档：<https://datatracker.ietf.org/doc/html/rfc2616>，但是因为它 是英文的，对很多四六级还没过的同学不是很友好（大家应该都比我英语好），我给大家推荐一本书叫《图解 HTTP》，并且给出它的电子版链接：<https://www.kancloud.cn/spirit-ling/http-study>，我们之后就参考这个文档就可以了。

2.5 Socket

Socket 也就是套接字，操作系统负责实现套接字，会提供一系列 API 供运行在上面的应用来使用。因此，在 Windows 里和 Linux 里的 Socket API 就不同，这也是我们一开始说请大家都统一在 Windows 上做实验的初衷。当然事实上在 Linux 上也有 Socket，而且实现地更优雅，有兴趣的同学们欢迎也在 Linux 上实现本文的实验，并尝试用条件编译为代码添加多种操作系统的编译支持。

在 Windows 中采用 WinSock（Windows Sockets）进行套接字编程，它提供了两种协议，分别是 TCP 和 UDP。我们都知道 HTTP 协议基于 TCP，因此我们将采用 TCP 进行开发。

三、理解 HTTP 协议

3.1 抓包观察

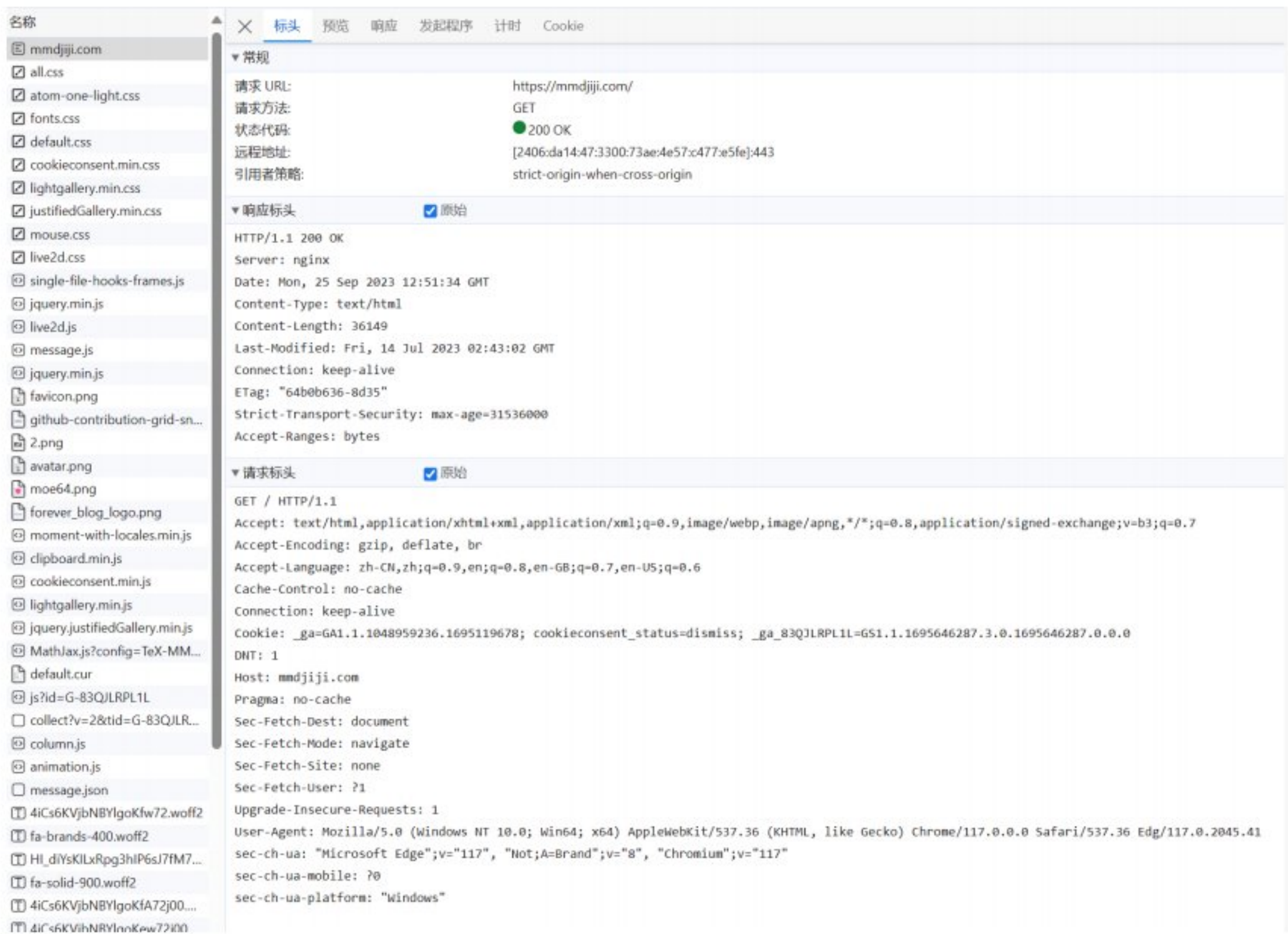


图 4

3.2 URL

URL，统一资源定位符，实际上就是网址，它的一般结构如下：

scheme://host:port/path?query#fragment

1. **scheme** (协议)：协议是 URL 的开头部分，通常指示了要使用的通信协议。它通常是小写字母，并且后跟一个冒号。常见的协议包括：
 - a) http：用于超文本传输协议，通常用于 Web 页面；
 - b) https：用于加密的 HTTP 版本，更安全；
 - c) ftp：用于文件传输协议，用于文件上传和下载；
 - d) mailto：用于电子邮件地址；
 - e) file：用于本地文件系统路径。
2. **host** (主机)：主机部分标识了资源所在的服务器或计算机的域名或 IP 地址。这通常是必需的，并且用于确定资源的位置。例如，www.example.com 或 192.168.1.1。
3. **port** (端口)：端口部分是可选的，它指定了要连接到主机的端口号。如果未明确指定端口号，将使用默认端口号。例如，HTTP 通常使用端口 80，HTTPS 通常使用端口 443。
4. **path** (路径)：路径部分指定了服务器上资源的具体位置或路径。它可以是目录路径或文件路径，以斜杠分隔。例如，/images/logo.png 表示位于根目录下的图像文件。
5. **query** (查询参数)：查询部分是可选的，它用于向服务器传递参数或数据。查询参数通常以问号 (?) 开头，多个参数之间以和号 (&) 分隔。例如，?page=1&search=keyword 可以用于指定页面和搜索关键字。
6. **fragment** (片段标识符)：片段部分也是可选的，通常用于指定资源内部的特定位置或锚点。它以井号 (#) 开头，后跟标识符。例如，#section2 可以用于直接跳转到页面中的第二部分。

3.3 请求头结构

Method URL HTTP/Version

Header1: Value1

Header2: Value2

...

1. **Method** (方法)：这是 HTTP 请求的方法，通常是 HTTP 动词，指示服务器应该执行的操作。常见的 HTTP 方法包括：
 - a) GET：请求获取指定资源；
 - b) POST：向指定资源提交数据进行处理（常用于表单提交）；
 - c) PUT：请求服务器存储一个资源，覆盖已存在的资源；
 - d) DELETE：请求服务器删除指定资源；
 - e) HEAD：类似于 GET 请求，但只返回响应头信息，不返回实际数据。
2. **URL**：这是请求的目标 URL（统一资源定位符），表示要访问的资源的位置。它包括协议、主机、端口、路径以及查询参数和片段标识符等信息。
3. **HTTP/Version**：这是 HTTP 协议的版本号，指示客户端所使用的 HTTP 协议版本。常见的版本包括 HTTP/1.1 和 HTTP/2。
4. **Headers** (头部字段)：这是一个或多个 HTTP 头部字段的集合，每个字段包括一个

名称和一个值，用冒号分隔。头部字段提供了关于请求的额外信息。常见的请求头部字段包括：

- a) Host: 指定请求的目标主机；
- b) User-Agent: 包含客户端的用户代理信息，通常表示浏览器或应用程序的标识；
- c) Content-Type: 指定请求主体的内容类型（仅适用于 POST 等请求方法）；
- d) Accept: 指定客户端可接受的响应内容类型；
- e) Authorization: 包含身份验证信息，用于访问受保护的资源；
- f) Cookie: 包含客户端的 Cookie 信息。

3.4 响应头结构

HTTP/Version Status Code Reason Phrase
Header1: Value1
Header2: Value2
...

1. **HTTP/Version** (HTTP 协议版本): 这是 HTTP 协议的版本号，表示服务器使用的 HTTP 协议版本。常见的版本包括 HTTP/1.1 和 HTTP/2。
2. **Status Code** (状态码): 状态码是一个三位数字，用于表示服务器对请求的处理结果。每个状态码有特定的含义，例如：
 - a) 200 OK: 请求成功，服务器正常响应；
 - b) 404 Not Found: 请求的资源未找到；
 - c) 500 Internal Server Error: 服务器内部错误。
3. **Reason Phrase** (状态码原因短语): 这是与状态码相关联的可选文本描述，提供了对状态码的更详细说明。通常，客户端可以忽略这个文本，因为状态码已经提供了足够的信息。
4. **Headers** (头部字段): 这是一个或多个 HTTP 头部字段的集合，每个字段包括一个名称和一个值，用冒号分隔。头部字段提供了关于响应的额外信息。常见的响应头部字段包括：
 - a) Content-Type: 指定响应主体的内容类型，告诉客户端如何解析响应数据；
 - b) Content-Length: 指定响应主体的长度（以字节为单位）；
 - c) Location: 在重定向响应中，指定客户端应该访问的新位置；
 - d) Server: 包含服务器的信息，通常是服务器软件的名称和版本号；
 - e) Set-Cookie: 用于设置 Cookie，以在客户端上跟踪会话状态。

3.5 问题转化

在进行 HTTP 请求时，浏览器就相当于客户端，我们要编写的就是服务端。服务端应当绑定一个端口进行监听，对于外来请求（识别到请求头、请求体），作出响应（发送响应头、响应体）。

实现 HTTP 协议的问题到这里就转化为了一个**字符串处理问题**，我们只需要实现两个部分的算法：

- i. 解析请求头；
- ii. 生成响应头。

当然，这也并不只是字符串处理，实际过程中还涉及到磁盘 IO，你解析到了用户要访问 index.html 的请求，自然要在网站目录下去读取 index.html 的内容并作为响应体发送给用户。

通常这类问题我们应该遵循《编译原理》中的词法分析、语法分析等方法来进行解析，但只是实现一个小型 HTTP 服务器无需大费周章，我们可以编写一个简单的状态机来实现这件事，甚至用字符串的模式匹配就能上手。

四、实验准备

4.1 先把 TCP Socket 跑起来

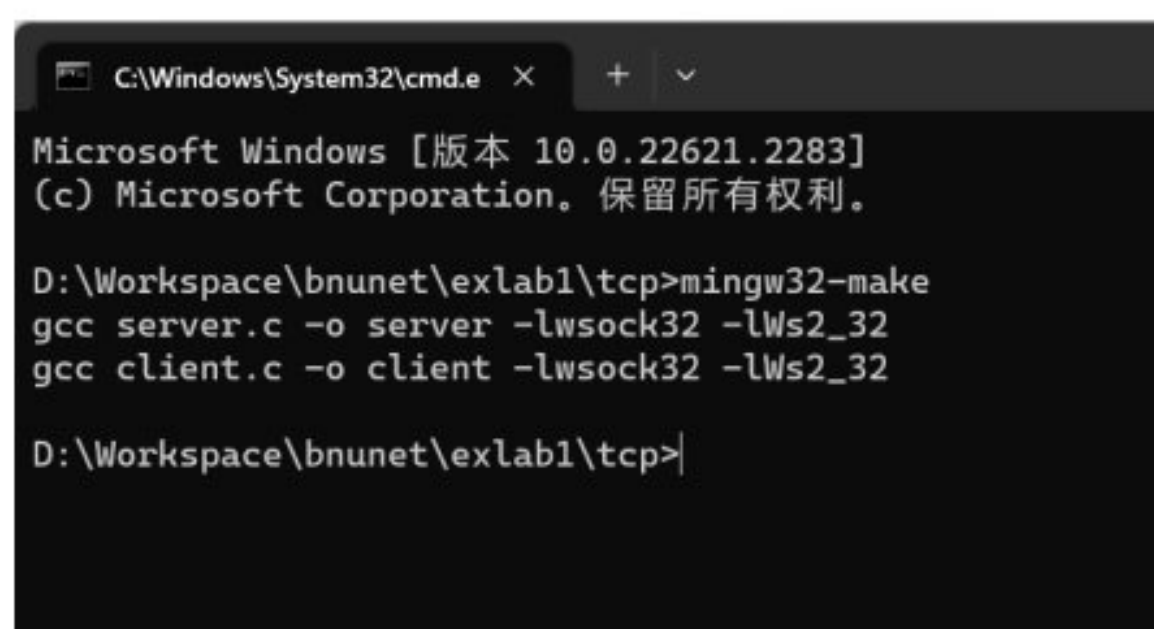
使用 Git，克隆本次实验所需的代码：

```
git clone https://github.com/bnunet/exlab1
```

如果上述代码较慢，可将 github 替换为 gitee 再进行尝试。

打开 tcp 文件夹，用 cmd 打开并输入 **mingw32-make** 指令以编译本项目（图 5）。

再在上述路径打开一个 cmd，输入 **server**，打开服务端（图 6）。再在上述路径打开一个 cmd，输入 **client 127.0.0.1**，表示打开客户端（图 7），连接到本地。在打开之后，客户端发送消息并立即退出，此时再查看服务端（图 8），接收到了消息，也退出了。这就表明完成了一次 TCP 通信。

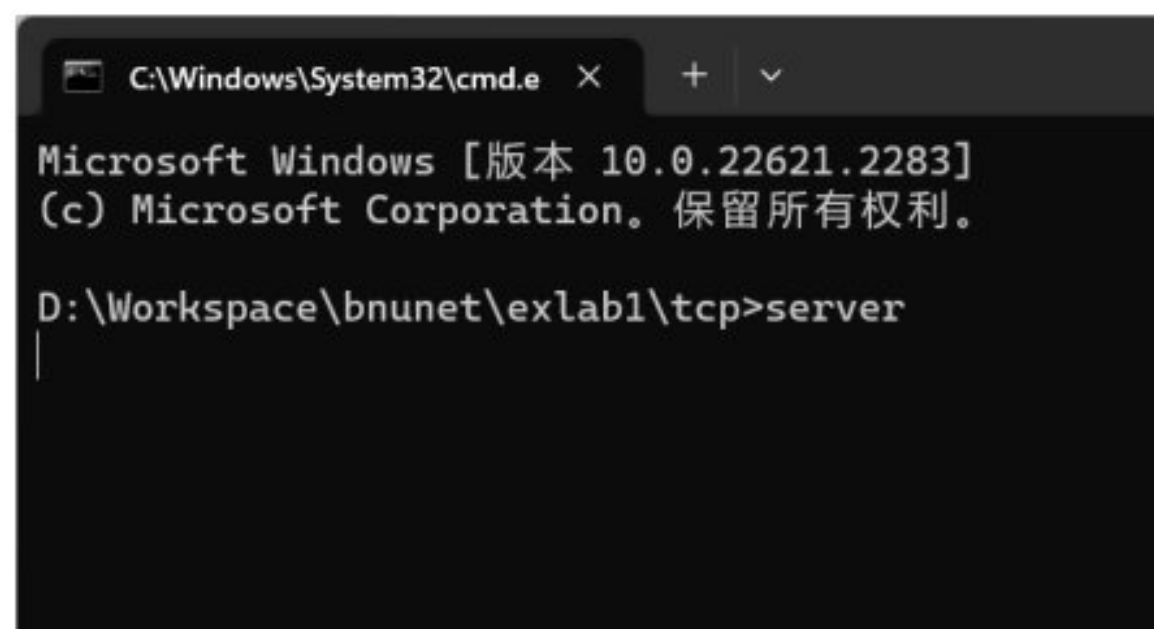


```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [版本 10.0.22621.2283]
(c) Microsoft Corporation. 保留所有权利。

D:\Workspace\bnunet\exlab1\tcp>mingw32-make
gcc server.c -o server -lws2_32
gcc client.c -o client -lws2_32

D:\Workspace\bnunet\exlab1\tcp>
```

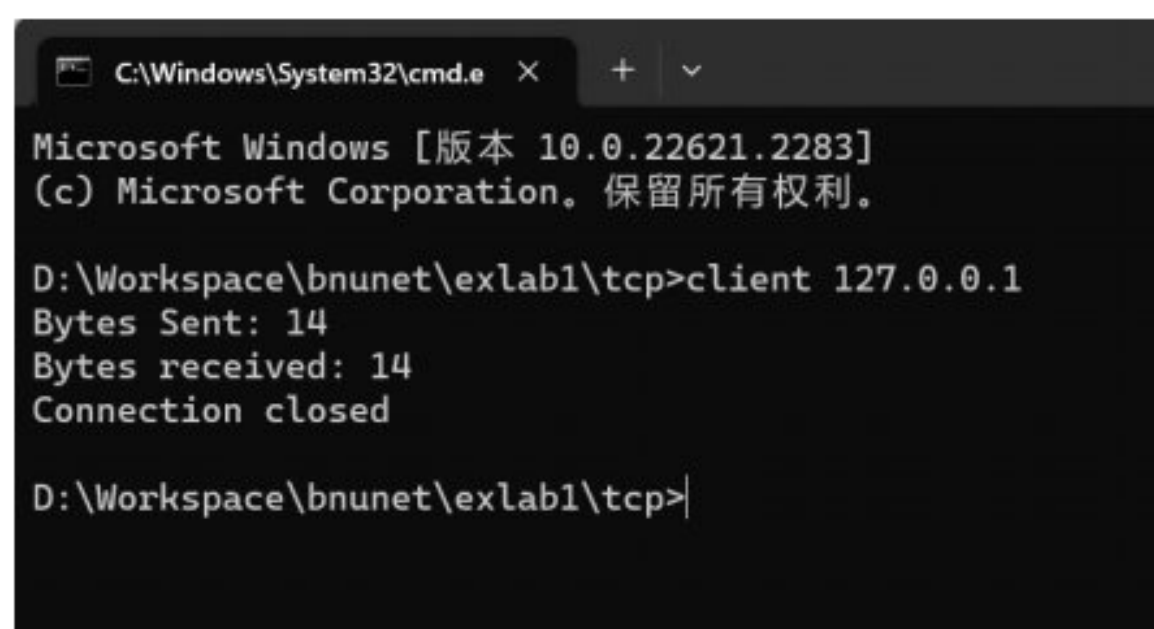
图 5



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [版本 10.0.22621.2283]
(c) Microsoft Corporation. 保留所有权利。

D:\Workspace\bnunet\exlab1\tcp>server
```

图 6

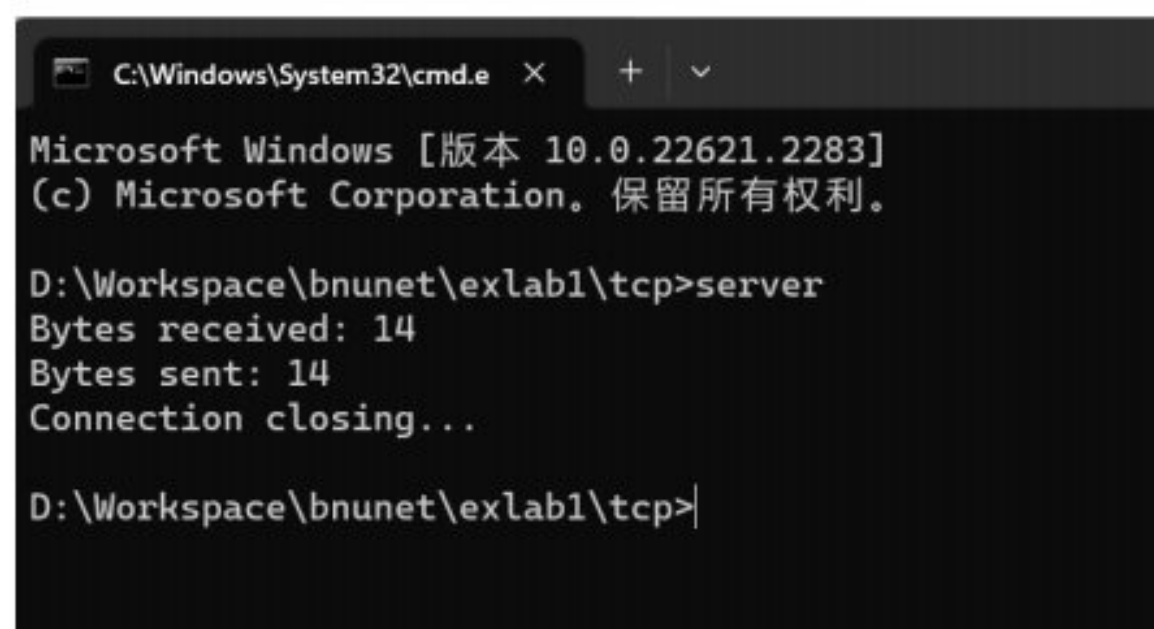


```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [版本 10.0.22621.2283]
(c) Microsoft Corporation. 保留所有权利。

D:\Workspace\bnunet\exlab1\tcp>client 127.0.0.1
Bytes Sent: 14
Bytes received: 14
Connection closed

D:\Workspace\bnunet\exlab1\tcp>
```

图 7



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [版本 10.0.22621.2283]
(c) Microsoft Corporation. 保留所有权利。

D:\Workspace\bnunet\exlab1\tcp>server
Bytes received: 14
Bytes sent: 14
Connection closing...

D:\Workspace\bnunet\exlab1\tcp>
```

图 8

这里给你的小任务是阅读并尝试理解这段代码，这段代码出自微软的官方文档：

<https://learn.microsoft.com/en-us/windows/win32/winsock/finished-server-and-client-code>

4.2 实现一个 HTTP 服务器

还是本次实验的代码，如果 4.1 中已经 clone 过则无需再次 clone，打开 src 文件夹。

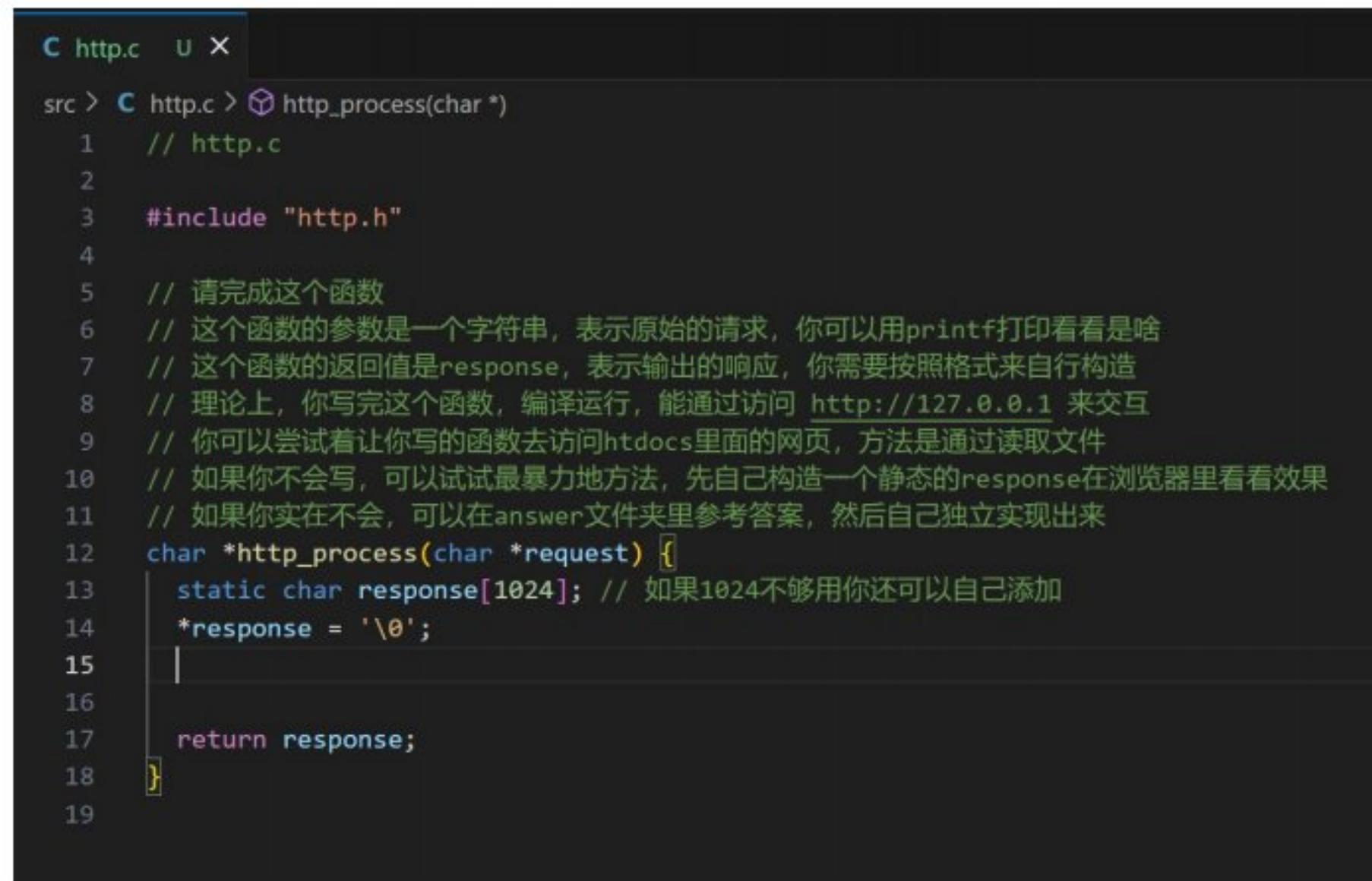
```
git clone https://github.com/bnunet/exlab1
```

你需要做的是实现 `http.c` 中的 `http_process` 函数，以完成一个最小的 `http/1.1` 的服务端，你可以先去研究实现 `200 OK` 的响应，再去研究实现 `404 Not Found` 的响应。都完成后，如果学有余力，你还可以研究怎么实现 `403 Forbidden`。

最终实现的版本至少包含 `200 OK` 和 `404 Not Found` 两个响应。

编译的命令还是 `mingw32-make`，运行的命令是 `exlab1`。

如果实在不会可以参考 `answer` 文件夹的内容。



```
C http.c U X
src > C http.c > http_process(char *)
1 // http.c
2
3 #include "http.h"
4
5 // 请完成这个函数
6 // 这个函数的参数是一个字符串，表示原始的请求，你可以用printf打印看看是啥
7 // 这个函数的返回值是response，表示输出的响应，你需要按照格式来自行构造
8 // 理论上，你写完这个函数，编译运行，能通过访问 http://127.0.0.1 来交互
9 // 你可以尝试着让你写的函数去访问htdocs里面的网页，方法是通过读取文件
10 // 如果你不会写，可以试试最暴力地方法，先自己构造一个静态的response在浏览器里看看效果
11 // 如果你实在不会，可以在answer文件夹里参考答案，然后自己独立实现出来
12 char *http_process(char *request) {
13     static char response[1024]; // 如果1024不够用你还可以自己添加
14     *response = '\0';
15
16
17     return response;
18 }
19
```

图 9

五、实验报告的提交

提交链接: <https://pan.bnu.edu.cn/l/J1Lcsj>

命名方法: ExLab1-姓名-学号.docx