Lab2: Your Own HTTP Server

Some materials are from Homework 2 of CS162 2019 at UC Berkeley. Thanks to CS162!

Enter in the folder you have cloned from our lab git repo, and pull the latest commit.

`git pull`

You can find this lab2's instruction in `Lab2/README.md`

All materials of lab2 are in folder `Lab2/`

1. Overview 概述

Implement an HTTP server from scratch by your own, using network programming knowledges learned from our class. Also, try to use high concurrency programming skills learned from the class to guarantee the web server's performance.

使用从我们课堂学到的网络编程知识，自己从头开始实现HTTP服务器。另外，尝试使用从课堂中学习的高并发编程技能来保证web服务器的性能。

Goals

- Practice basic network programming skills, such as using socket API, parsing packets;
- Get familiar with robust and high-performance concurrent programming.

练习基本的网络编程技巧，比如使用socket API，解析数据包
熟悉健壮、高性能的并发编程。

2. Background

2.1 Hypertext Transport Protocol

The Hypertext Transport Protocol (HTTP) is the most commonly used application protocol on the Internet today. Like many network protocols, HTTP uses a client-server model. An HTTP client opens a network connection to an HTTP server and sends an HTTP request message. Then, the server replies with an HTTP response message, which usually contains some resource (file, text, binary data) that was requested by the client. We briefly introduce the HTTP message format and structure in this section for your convenience. Detailed specification of HTTP/1.1 can be found in RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.

超文本传输协议（HTTP）是当今互联网上最常用的应用协议。与许多网络协议一样，HTTP使用客户机-服务器模型。HTTP客户端打开到HTTP服务器的网络连接并发送HTTP请求消息。然后，服务器用HTTP响应消息进行响应，该消息通常包含客户端请求的某些资源（文件、文本、二进制数据）。为了您的方便，我们将在本节简要介绍HTTP消息的格式和结构。HTTP/1.1的详细规范可以在RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.

2.2 HTTP Messages

HTTP messages are simple, formatted blocks of data. All HTTP messages fall into two types: **request** messages and **response** messages. Request messages request an action from a web server. Response messages carry results of a request back to a client. Both request and response messages have the same basic message structure.

HTTP消息是简单的格式化数据块。所有HTTP消息分为两种类型：请求消息和响应消息。请求消息请求来自web服务器的操作。响应消息将请求的结果传回客户端。请求和响应消息都具有相同的基本消息结构。

2.2.1 Message Format

HTTP request and response messages consist of 3 components(组件)：

- a start line describing the message，描述消息的起始行
- a block of headers containing attributes, 包含属性的头块
- and an optional body containing data.包含数据的可选实体

Each component has the format as following

2.2.1.1 Start Line

All HTTP messages begin with a start line. The start line for a request message says *what to do*. The start line for a response message says *what happened*.
所有HTTP消息都以起始行开头。请求消息的起始行说明要做什么。响应消息的起始行说明发生了什么。

Specifically, the start line is also called ***Request line*** in *Request messages* and ***Response line*** in *Response messages*.
具体地说，在请求消息中，起始行也称为请求行，在响应消息中称为响应行。

1. **Request line:** The request line contains a method describing what operation the server should perform and a request URL describing the resource on which to perform the method. The request line also includes an HTTP version tells the server what dialect of HTTP the client is speaking. All of these fields are separated by whitespace.
   请求行：请求行包含描述服务器应执行的操作的方法，以及描述要在其上执行该方法的资源的请求URL。请求行还包括一个HTTP版本，它告诉服务器客户端所说的HTTP方言是什么。所有这些字段都用空格分隔。

Example of request line:

```
GET /index.html HTTP/1.0
```

2. **Response line:** The response line contains the HTTP version that the response message is using, a numeric status code, and a textual reason phrase describing the status of the operation.
   响应行：响应行包含响应消息正在使用的HTTP版本、数字状态代码和描述操作状态的文本原因短语。

Example of response line:

```
HTTP/1.0 200 OK
```

2.2.1.2 Header

Following the start line comes a list of zero, one, or many HTTP header fields. HTTP header fields add additional information to request and response messages. They are basically just lists of name/value pairs. Each HTTP header has a simple syntax: a name, followed by a colon (:), followed by optional whitespace, followed by the field value, followed by a CRLF.
在开始行后面是一个包含零个、一个或多个HTTP头字段的列表。HTTP头字段向请求和响应消息添加附

加信息。它们基本上只是名称/值对的列表。每个HTTP头都有一个简单的语法：名称，后跟冒号（：），后跟可选空白，后跟字段值，后跟CRLF

HTTP headers are classified into: General headers, Request headers, Response headers, Entity headers and Extension headers. Note that request-header fields are different from the response-header fields. We will not introduce those fields in details and you are not required to implement in this lab. You can find them in RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.
HTTP头分为：常规头、请求头、响应头、实体头和扩展头。请注意，请求头字段与响应头字段不同。我们不会详细介绍这些字段，也不要求您在此实验室中实现这些字段。您可以在RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.

Example of headers in a request:

```
Host: 127.0.0.1:8888
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
                                    // CRLF
```

Example of headers in a response:

```
Server: Guo's Web Server
Content-length: 248
Content-type: text/html
                                  // CRLF
```

2.2.1.3 Entity Body

The third part of an HTTP message is the optional entity body. Entity bodies are the payload of HTTP messages. They are the things that HTTP was designed to transmit.
HTTP消息的第三部分是可选实体体。实体体是HTTP消息的有效负载。它们是HTTP设计用来传输的东西。

HTTP messages can carry many kinds of digital data: images, video, HTML documents, software applications, credit card transactions, electronic mail, and so on.
HTTP消息可以承载多种数字数据：图像、视频、HTML文档、软件应用程序、信用卡交易、电子邮件等。

Example of entity body:

```
<html><head>
<title>CSO6142</title>
</head><body>
<h1>CSO6142</h1>
<p>Welcome to Cloud Computing Course.<br />
</p>
<hr>
<address>Http Server at ip-127-0-0-1 Port 8888</address>
</body></html>
```

2.2.2 Structure of HTTP Request

A HTTP request message contains an HTTP request line (containing a method, a query string, and the HTTP protocol version), zero or more HTTP header lines and a blank line (i.e. a CRLF).
HTTP请求消息包含HTTP请求行（包含方法、查询字符串和HTTP协议版本）、零个或多个HTTP头行和空行（即CRLF）。

Example of HTTP request message:

```
GET /index.html HTTP/1.0
Host: 127.0.0.1:8888
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
                                    // CRLF
```

2.2.3 Structure of HTTP Response

A HTTP response message contains an HTTP response status line (containing the HTTP protocol version, the status code, and a description of the status code), zero or more HTTP header lines, a blank line (i.e. a CRLF by itself) and the content requested by the HTTP request.
HTTP响应消息包含HTTP响应状态行（包含HTTP协议版本、状态代码和状态代码的描述）、零个或多个HTTP头行、空行（即CRLF本身）和HTTP请求请求所请求的内容。

Example of HTTP response message:

```
HTTP/1.0 200 OK
Server: Tiny Web Server
Content-length: 248
Content-type: text/html
                                    // CRLF
<html><head>
<title>CS06142</title>
</head><body>
<h1>CS06142</h1>
<p>Welcome to Cloud Computing Course.<br />
</p>
<hr>
<address>Http Server at ip-127-0-0-1 Port 8888</address>
</body></html>
```

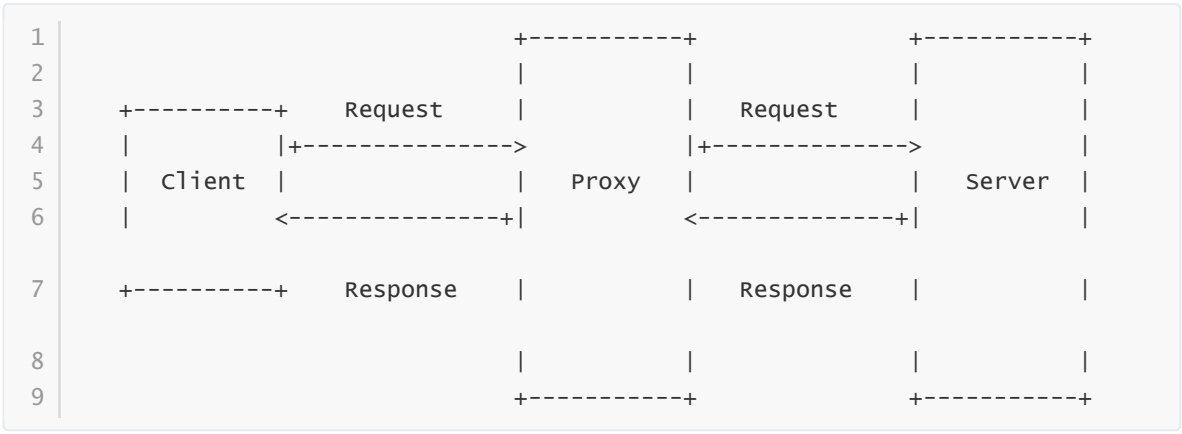
2.3 HTTP Proxy

HTTP proxy servers are intermediaries. Proxies sit between clients and servers and act as "middlemen", shuffling HTTP messages back and forth between the parties.
HTTP代理服务器是中介。代理位于客户机和服务器之间，充当“中间人”，在双方之间来回传递HTTP消息。

HTTP proxy servers are middlemen that fulfill transactions on the client's behalf. Without a HTTP proxy, HTTP clients talk directly to HTTP servers. With a HTTP proxy, the client instead talks to the proxy, which itself communicates with the server on the client's behalf.
HTTP代理服务器是代表客户完成交易的中间人。没有HTTP代理，HTTP客户端直接与HTTP服务器对话。使用HTTP代理，客户机将与代理进行对话，代理本身代表客户机与服务器进行通信。

HTTP proxy servers are both web servers and web clients. Because HTTP clients send request messages to proxies, the proxy server must properly handle the requests and the connections and return responses, just like a web server. At the same time, the proxy itself sends requests to servers, so it must also behave like a correct HTTP client, sending requests and receiving responses.
HTTP代理服务器既是web服务器又是web客户端。因为HTTP客户端向代理发送请求消息，代理服务器必须像web服务器一样正确地处理请求、连接和返回响应。同时，代理本身向服务器发送请求，因此它也必须像一个正确的HTTP客户端一样，发送请求并接收

The working pattern of HTTP proxy is shown in the following figure:



3. Your Lab Task

3.1 Implement your own HTTP Server

In this Lab, we won't provide any basic code. So, you should implement a HTTP server from scratch which satisfies the following requirements:
在这个实验里，我们不会提供任何基本代码。因此，您应该从头开始实现满足以下要求的HTTP服务器

3.1.1 HTTP Server Outline HTTP服务器大纲

From a network standpoint, your HTTP server should implement the following:
从网络的角度来看，HTTP服务器应该实现以下功能:

1. Create a listening socket and bind it to a port
2. Wait a client to connect to the port
3. Accept the client and obtain a new connection socket
4. Read in and parse the HTTP request
5. Start delivering services: 1) Handle HTTP GET/POST request and return an error message if an error occur. 2) Proxy the request to another HTTP server (optional for advanced version).
   1、创建监听套接字并将其绑定到端口
   2、等待客户端连接到端口
   3、接受客户端并获取新的连接套接字
   4、读入并分析HTTP请求
   5、开始交付服务：

1）处理HTTP GET/POST请求，并在发生错误时返回错误消息。
2）将请求代理到另一个HTTP服务器（对于高级版本是可选的）。

The server will be in either non-proxy mode or proxy mode (we have introduced the proxy in section `2.3`). It does not do both things at the same time.
服务器将处于非代理模式或代理模式（我们在第2.3节中介绍了代理）。它不能同时做两件事。

3.1.2 Handle HTTP request

In this Lab, **you just need to implement the GET method and the POST method in your HTTP server**. That is to say, if your HTTP server receive a HTTP request but the request method is neither GET nor POST. The HTTP server just need to return a 501 Not Implemented error message (a response message with Response line having status code to be 501, see `2.2`).
在这个实验中，您只需要在HTTP服务器中实现GET方法和POST方法。也就是说，如果HTTP服务器接收到HTTP请求，但请求方法既不是GET也不是POST。HTTP服务器只需要返回一条501未实现的错误消息（响应行状态代码为501的响应消息，请参阅2.2）。

There is no need to handle the header line(s) in the request (but you need to recognize it so that you will not mix it with the next request's start line). Also, you are free to fill any (or zero) header line(s) in the HTTP response.
不需要处理请求中的标题行（但您需要识别它，以便不会将其与下一个请求的起始行混合）。此外，您可以随意填充HTTP响应中的任何（或零）标题行。

3.1.2.1 Handle HTTP GET request

The HTTP server should be able to handle HTTP GET requests for html pages.
HTTP服务器应该能够处理html页面的HTTP GET请求。

1. If the HTTP request's path corresponds to a html page file, respond with a 200 OK and the full contents of the file. For example, if GET /index.html is requested, and a file named index.html exists in the files directory. You should also be able to handle requests to files in subdirectories of the files directory (e.g. GET /images/hero.jpg).
   如果HTTP请求的路径对应于html页面文件，则使用200 ok和文件的全部内容进行响应。例如，如果请求GET /index.html，并且文件目录中存在名为index.html的文件。您还应该能够处理对files目录的子目录中的文件的请求（例如GET/images/hero.jpg）。
2. If the HTTP request's path corresponds to a directory and the directory contains an `index.html` file, respond with a 200 OK and the full contents of the index.html file in that folder.
   如果HTTP请求的路径对应于一个目录，并且该目录包含index.html文件，则使用200 OK和该文件夹中index.html文件的全部内容进行响应。
3. If the requested page file does not exist, or the requested directory does not contain an `index.html` file, return a 404 Not Found response (the HTTP body is optional).
   如果请求的页面文件不存在，或者请求的目录不包含index.html文件，则返回404 not Found响应（HTTP正文是可选的）。

3.1.2.2 Handle HTTP POST request

The HTTP server should be able to handle HTTP POST requests. In this lab, the way of handle HTTP POST is very simple.
HTTP服务器应该能够处理HTTP POST请求。在这个实验中，处理HTTP POST的方法非常简单。

1. You should construct an HTTP POST request (see section `3.1.7.2`) that contains 2 keys: "Name" and "ID" (please fill in your name and student number respectively), and send the POST request to `/Post_show` (i.e. `http://127.0.0.1:8888/Post_show` if server's IP is `127.0.0.1` and service port is `8888`).
   您应该构造一个包含两个键：“Name”和“ID”（请分别填写您的姓名和学号）的HTTP POST请求

（请参见第3.1.7.2节），并将POST请求发送到/POST_show（即，如果服务器的IP为127.0.0.1，服务端口为8888，则HTTP://127.0.0.1:8888/POST_show）

Then, if the HTTP server receive this POST request (the request URL is `/Post_show` and the keys are "Name" and "ID"), respond with a 200 OK and echo the "Name"-"ID" pairs you have sent (see section `3.1.7.2`).
然后，如果HTTP服务器接收到这个POST请求（请求URL是/POST-show，键是“Name”和“ID”），则用200ok响应并回显您发送的“Name”-“ID”对（参见第3.1.7.2节）。

2. Otherwise (i.e. the request URL is not `/Post_show` or the keys are not "Name" and "ID"), return a 404 Not Found response message.
   否则（即请求URL不是/Post\u show或密钥不是“Name”和“ID”），返回404 not Found响应消息。

3.1.2.3 Other request

Just return 501 Not Implemented error message for other request method (e.g. DELETE, PUT, etc. see section `3.1.7.3`).
对于其他请求方法，只需返回501未实现的错误消息（例如删除、放置等，请参阅第3.1.7.3节）。

3.1.3 Implement a proxy server (optional)实现代理服务器（可选）

Enable your server to proxy HTTP requests to another HTTP server and forward the responses to the clients.
使您的服务器能够将HTTP请求代理到另一个HTTP服务器并将响应转发到客户端

1. You should use the value of the `--proxy` command line argument, which contains the address and port number of the upstream HTTP server.
   您应该使用--proxy命令行参数的值，该参数包含上游HTTP服务器的地址和端口号。
2. Your proxy server should wait for new data on both sockets (the HTTP client fd, and the upstream HTTP server fd). When data arrives, you should immediately read it to a buffer and then write it to the other socket. You are essentially maintaining 2-way communication between the HTTP client and the upstream HTTP server. Note that your proxy must support multiple requests/responses.
   代理服务器应该在两个套接字（HTTP客户端fd和上游HTTP服务器fd）上等待新数据。当数据到达时，您应该立即将其读到一个缓冲区，然后将其写入另一个套接字。实际上，您在维护HTTP客户端和上游HTTP服务器之间的双向通信。请注意，代理必须支持多个请求/响应。
3. If either of the sockets closes, communication cannot continue, so you should close the other socket and exit the child process.
   如果其中一个套接字关闭，则无法继续通信，因此应关闭另一个套接字并退出子进程。

Hints: 1) This is more tricky than writing to a file or reading from stdin, since you do not know which side of the 2-way stream will write data first, or whether they will write more data after receiving a response. 2) You should again use threads for this task. For example, consider using two threads to facilitate the two-way communication, one from A to B and the other from B to A.
提示：1）这比写入文件或从stdin读取要复杂得多，因为您不知道双向流的哪一边将首先写入数据，或者在接收到响应后它们是否将写入更多数据。2） 您应该再次使用线程执行此任务。例如，考虑使用两个线程来促进双向通信，一个从A到B，另一个从B到A。

3.1.4 Use multi-thread to increase concurrency使用多线程来提高并发性

Your HTTP server should use multiple threads to handle as many concurrent clients' requests as possible. You have at least the following three options to architect your multi-thread server:
您的HTTP服务器应该使用多线程来处理尽可能多的并发客户端请求。您至少有以下三个选项来构建多线程服务器：

- **On-demand threads**. You can can create a new thread whenever a new client comes in, and use that thread to handle all that clients' task, including parsing the HTTP request, fetching page files, and sending response. The thread can be destroyed after that client finishes, e.g, detect through TCP `recv()`. However,it may not be easy to detect client finish in the HTTP layer.
  按需线程：您可以在新客户机进入时创建一个新线程，并使用该线程处理所有客户机的任务，包括解析HTTP请求、获取页面文件和发送响应。在客户端完成后，可以销毁线程，例如，通过TCP recv（）进行检测。但是，在HTTP层中检测客户端完成可能并不容易。
- **A pool of always-on threads**. You can use a fixed-sized thread pool in your HTTP server program for handling multiple client requests concurrently. If there are no tasks, those threads are in a waiting state. If a new client comes in, assign a thread to handle the client's request and send response to it. If the assigned thread is busy, you can use a work queue to buffer the request, and let the thread process it later.
  一个总有线程的池：您可以在HTTP服务器程序中使用固定大小的线程池来同时处理多个客户端请求。如果没有任务，则这些线程处于等待状态。如果新的客户机进入，则分配一个线程来处理客户机的请求并向其发送响应。如果分配的线程正忙，可以使用工作队列缓冲请求，并让线程稍后处理它。
- **Combined**. Combine above two styles together. For example, you can use thread pool to receive request and send response, and use on-demand threads to fetch large page files.
  合并：将以上两种风格结合在一起。例如，可以使用线程池接收请求和发送响应，并使用按需线程获取大型页文件。
  Feel free to choose any one from the above three, or use other multi-thread architecture that you think is cool.
  您可以自由选择以上三种架构中的任何一种，或者使用您认为很酷的其他多线程架构

3.1.5 Specify arguments 指定参数

Your program should enable long options to accept arguments and specify those arguments during start. They are `--ip`, `--port`, and `--proxy` (optional).
您的程序应启用长选项来接受参数，并在启动期间指定这些参数。它们是--ip、--port和--proxy（可选）。

1. `--ip` —— Specify the server IP address. 指定服务器ip地址
2. `--port` —— Selects which port the HTTP server listens on for incoming connections. 选择HTTP服务器监听传入连接的端口。
3. `--proxy` —— Selects an “upstream” HTTP server to proxy. The argument can have a port number after a colon (e.g. `https://www.cs06142.com:80`). If a port number is not specified, port 80 is the default.
   选择要代理的“上游”HTTP服务器。参数可以在冒号后有端口号（例如https://www.CS06142.com:80）。如果未指定端口号，则默认为端口80。

If you have no idea about *long options*, you can read this. And you may need to use some functions like `getopt_long()`, `getopt_long_only()`, `getopt()` and so on. Check those function's usage with the `man` command.
如果你不知道长的选择，你可以读这个。您可能需要使用一些函数，如getopt_long（）、getopt_long_only（）、getopt（）等等。用man命令检查这些函数的用法。

3.1.6 Run your HTTP Server

Your program should be able to start at terminal. If your program is called *httpserver*, just typing:
你的HTTP服务器你的程序应该能够在终端启动。如果程序名为HTTP server，只需键入

in the non-proxy mode:
在非代理模式下：
`./httpserver --ip 127.0.0.1 --port 8888 --number-thread 8`

It means that your HTTP server's IP address is 127.0.0.1 and service port is 8888. The --number-thread indicates that there are 8 threads in the thread pool for handling multiple client request concurrently.
表示HTTP服务器的ip地址为127.0.0.1，服务端口为8888。--numberthread表示线程池中有8个线程用于同时处理多个客户端请求。

in the proxy mode:
在代理模式下：
`./httpserver --ip 127.0.0.1 --port 8888 --number-thread 8 --proxy https://www.cs06142.com:80`

It means that this is an HTTP proxy. This proxy's IP address is 127.0.0.1 and service port is 8888. And the proxy has a thread pool with 8 threads. The --proxy indicates that the "upstream" HTTP server is `https://www.cs06142.com:80`. So, if you send a request message to this proxy (i.e. `127.0.0.1:8888`), it will forward this request message to the "upstream" HTTP server (i.e. `https://www.cs06142.com:80`) and forward the response message to the client.
表示这是一个HTTP代理。此代理的IP地址为127.0.0.1，服务端口为8888。代理有一个有8个线程的线程池。--proxy表示“上游”HTTP服务器是https://www.CS06142.COM:80。因此，如果您向此代理发送请求消息（即127.0.0.1:8888），它将把此请求消息转发到“上游”HTTP服务器（即https://www.CS06142.com:80），并将响应消息转发到客户端。

When you run the command above, your HTTP server should run correctly.
当您运行上面的命令时，您的HTTP服务器应该正常运行。

3.1.7 Accessing Your HTTP Server

3.1.7.1 Using GET method

1. You can check that your HTTP server works by opening your web browser and going to the appropriate URL. [Note] IP 127.0.0.1 refers to the IP of local host. So you can use 127.0.0.1 to test your HTTP server on the same local machine.
   打开web浏览器并转到适当的URL，可以检查HTTP服务器是否正常工作。
   [注]IP 127.0.0.1是指本地主机的IP。因此，可以使用127.0.0.1在同一台本地计算机上测试HTTP服务器。

For example:

2. You can also send HTTP requests with the curl program. An example of how to use curl is:
   您还可以使用curl程序发送HTTP请求。

`curl -i -X GET http://127.0.0.1:8888/index.html`

For example:

3. If the request page is non-existent, your HTTP server should return a 404 Not Found error message.
   如果请求页不存在，则HTTP服务器应返回404 Not Found错误消息。

For example:

3.1.7.2 Using POST method

1. You can check whether the POST method works by sending a HTTP request with the curl program. Typing the command at terminal:

可以通过使用curl程序发送HTTP请求来检查POST方法是否有效

```
curl -i -X POST --data 'Name=HNU&ID=CS06142' http://127.0.0.1:8888/Post_show
```

For example:

2. If the request URL is not `/Post_show` or the keys are not "Name" and "ID"), you will get a 404 Not Found error message.
   如果请求URL不是/Post_show，或者密钥不是“Name”和“ID”，则会收到404 not Found错误消息。

For example:

3. You can also construct a POST HTTP request and send the request to HTTP server using some browser plug-in tools.
   您还可以构造一个POST-HTTP请求，并使用一些浏览器插件工具将该请求发送到HTTP服务器。

3.1.7.3 Other method

The HTTP server will not handle HTTP requests except GET requests and POST requests.

If you send a HTTP DELETE (or PUT, HEAD, etc.) request to delete the specified resource, you will get a 501 Not Implemented error message:
除了GET请求和POST请求之外，HTTP服务器不会处理HTTP请求。如果发送HTTP DELETE（或PUT、HEAD等）请求以删除指定的资源，将收到501 Not Implemented错误消息：

3.1.8 Implementation requirements

3.1.8.1 Basic version 基础版本

Your program should complete all the **tasks described in section** `3.1.1~3.1.7` **except** `3.1.3`.

In the basic version, you have **only one request per TCP connection going on at the same time**. The client waits for response, and when it gets response, perhaps reuses the TCP connection for a new request (or use a new TCP connection). This is also what normal HTTP server supports.
除3.1.3外，您的程序应完成3.1.1~3.1.7节中描述的所有任务。在基本版本中，每个TCP连接只有一个请求同时进行。客户机等待响应，当它得到响应时，可能会对新请求重用TCP连接（或使用新的TCP连接）。这也是普通HTTP服务器所支持的。

3.1.8.2 Advanced version高级版本

Your program should complete all the **tasks described in section** `3.1.1~3.1.7` **including** `3.1.3`.

In the advanced version, **multiple http requests can be fired concurrently on one TCP connection**. This is also called HTTP pipelining which is supported by many real browsers and servers (such as Chrome). Note that HTTP requests that come from the same TCP connection should be responded in the same order. So take care the order problem when using complex multi-thread styles.
您的程序应完成3.1.1~3.1.7节中描述的所有任务，包括3.1.3节。在高级版本中，可以在一个TCP连接上同时触发多个http请求。这也被称为HTTP管道，它被许多真正的浏览器和服务器（如Chrome）所支持。请注意，来自同一个TCP连接的HTTP请求应以相同的顺序响应。所以在使用复杂的多线程样式时要注意顺序问题。

3.2 Finish a performance test report

Please test your code first, and commit a test report along with your lab code into your group's course github repo.
请先测试您的代码，并将测试报告和您的实验室代码一起提交到您组的课程github repo中：

The test report should describe the performance result under various testing conditions. Specifically, in your test report, you should at least contain the following two things:
测试报告应描述各种测试条件下的性能结果。具体来说，在测试报告中，至少应该包含以下两个内容：

1. Test how many HTTP request your server can process per second, when running on various server machine environments. For example, change the number of server CPU cores, enable/disable hyper-threading, etc.
   测试在不同的服务器机器环境中运行时，服务器每秒可以处理多少HTTP请求。例如，更改服务器CPU内核数、启用/禁用超线程等。
2. Test how many HTTP request your server can process per second, by varying the number of concurrent clients that send request to your server simultaneously. Do change the client's workload. For example, test when a client use new TCP connection for a new request, or when a client reuses old TCP connection for new requests. Moreover, if you implement the advanced version, try to change the number of out-bounding requests on the same client's TCP connection. You can write a simple client that send HTTP Get by your own (can run multiple client programs on the same machine to emulate multiple clients), or use some existing HTTP client testing tools such as ab - Apache HTTP server benchmarking tool.
   通过改变同时向服务器发送请求的并发客户端的数量，测试服务器每秒可以处理多少HTTP请求。一定要改变客户的工作量。例如，测试客户机何时对新请求使用新的TCP连接，或者客户机何时对新请求重用旧的TCP连接。此外，如果实现高级版本，请尝试更改同一客户端的TCP连接上的外边界请求数。您可以编写一个简单的客户端来自己发送HTTP Get（可以在同一台计算机上运行多个客户端程序来模拟多个客户端），或者使用一些现有的HTTP客户端测试工具，如ab-Apache HTTP server benchmarking工具。

[NOTE]: Be careful that clients may be the performance bottleneck. So you'd better use multiple machines when testing the performance. For example, you can run multiple client processes on three machines (of three group members), and run the server process on another machine (of the other group member). Moreover, the network can be the bottleneck too. You can estimate the performance limit according to the physical bandwidth of your network environment, and see if your implementation can reach the performance limit.
[注意]：注意客户端可能是性能瓶颈。所以在测试性能时最好使用多台机器。例如，可以在三台计算机（三个组成员）上运行多个客户端进程，并在另一台计算机（另一个组成员）上运行服务器进程。此外，网络也可能成为瓶颈。您可以根据网络环境的物理带宽来估计性能限制，并查看您的实现是否能够达到性能限制。

4. Lab submission

Please put all your code in folder `Lab2` and write a `Makefile` so that we **can compile your code in one single command** `make`. The compiled runnable executable binary should be named `httpserver` and located in folder `Lab2`. Please carefully following above rules so that TAs can automatically test your code!!!
请将您的所有代码放在文件夹Lab2中，并编写一个Makefile，以便我们可以在一个命令make中编译您的代码。编译后的可运行可执行二进制文件应命名为httpserver，并位于文件夹Lab2中。请认真遵守以上规则，助教会自动测试你的程式码！！！

Please submit your lab program and performance test report following the guidance in the Overall Lab Instructions (`../README.md`)
请按照总体实验室说明（../README.md）中的指导提交您的实验室计划和性能测试报告

5. Grading standards

1. You can get 23 points if you can: 1) finish all the requirements of the basic version, and 2) your performance test report has finished the two requirements described before. If you missed some parts, you will get part of the points depending how much you finished.
   如果可以的话，你可以得到23分：1）完成基本版本的所有需求，2）你的性能测试报告已经完成了前面描述的两个需求。如果你漏掉了一些部分，你将得到部分的分数取决于你完成了多少
2. You can get 25 points (full score) if you can: 1) finish all the requirements of the advanced version, and 2) your performance test report has finished the two requirements described before. If you missed some parts, you will get part of the points depending how much you finished.
   如果可以的话，你可以得到25分（满分）：1）完成高级版本的所有要求，2）你的性能测试报告已经完成了前面描述的两个要求。如果你漏掉了一些部分，你将得到部分的分数取决于你完成了多少。