1 概述

使用从我们的课上学到的网络编程知识，自己从零开始实现一个基于HTTP/1.1的HTTP服务器。此外，尝试使用从课堂上中学习的高并发编程技能来保证web服务器的性能。

目标:

实践基本的网络编程技能，如使用socket API，解析数据包；熟悉鲁棒和高性能并发编程。

2 背景

2.1 超文本传输协议

超文本传输协议(HTTP)是当今Internet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客户机打开到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一些资源(文件、文本、二进制数据)。为了方便您，我们将在本节简要介绍HTTP消息格式和结构。HTTP/1.1的详细规范可以在RFC 2616 -超文本传输协议

——HTTP/1.1中找到。

2.2 HTTP消息

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送回客户机。请求和响应消息都具有相同的基本消息结构。

2.2.1 消息格式

HTTP请求和响应消息由3个组件组成:

-描述信息的起始行start line

-包含属性的报头块，

-和一个可选的主体，包含数据。

每个组件的格式如下:

2.2.1.1 Start line

所有HTTP消息都以起始行开始。请求消息的起始行表示"要做什么"。响应消息的起始行表示"发生了什么"。

具体来说，起始行在请求消息中也称为请求行，在响应消息中称为响应行。

1. **请求行:** 请求行包含一个描述服务器应该执行什么操作的方法和一个描述执行该方法的资源的请求URL。请求行还包括一个HTTP版本，它告诉服务器客户机使用的是哪种HTTP方言。所有这些字段都用空格分隔。

请求行示例:

```
GET /index.html HTTP/1.1
```

2. **响应行:** 响应行包含响应消息使用的HTTP版本、一个数字状态代码和一个描述操作状态的文本原因短语。

响应行示例:

```
HTTP/1.1 200 OK
```

2.2.1.2 Header

开始行之后是一个包含0、1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:名称后面跟一个冒号(:)，后面跟一个可选的空格，后面跟一个字段值，后面跟一个CRLF。

HTTP报头分为:通用报头、请求报头、响应报头、实体报头和扩展报头。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，也不要求您在本实验室中实现它们。您可以在RFC 2616 -超文本传输协议- HTTP/1.1中找到它们。

请求中Header的例子:

```
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
```

响应中Header的例子:

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

2.2.1.3 Entity Body

HTTP消息的第三部分是可选的实体主体。实体实体是HTTP消息的有效负载。它们是HTTP被设计用来传输的东西。

HTTP消息可以携带多种数字数据:图像、视频、HTML文档、软件应用程序、信用卡交易、电子邮件等等。

实体主体示例:

```
<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 HTTP请求的结构

HTTP请求消息包含一个HTTP请求行(包含一个方法、一个查询字符串和HTTP协议版本)、零个或多个HTTP头行和一个空行(即CRLF)。

HTTP请求消息的例子:

```
GET /index.html HTTP/1.1
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 HTTP响应的结构

HTTP响应消息包含一个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、一个空行(即一个自身的CRLF)和HTTP请求请求的内容。

HTTP响应消息示例：

```
HTTP/1.1 200 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>
```

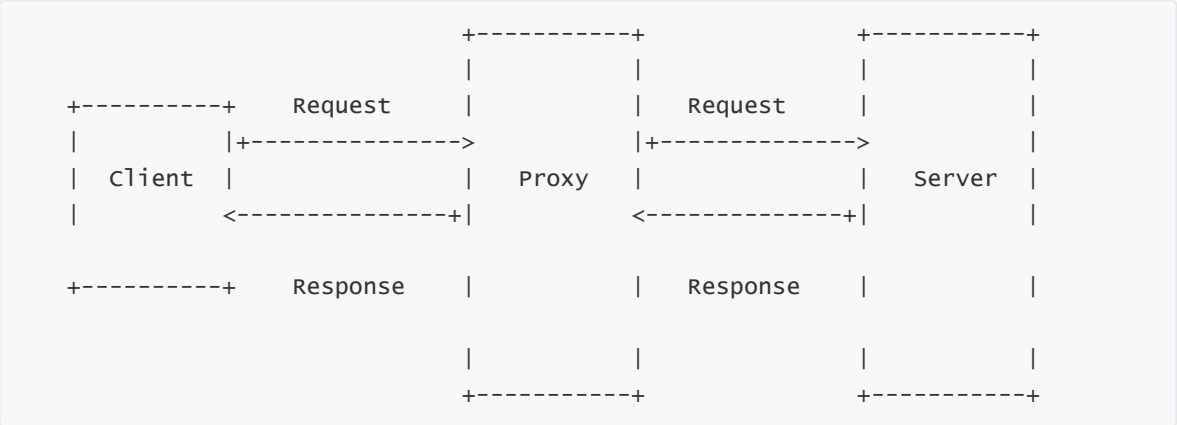
2.3 HTTP代理

HTTP代理服务器是中介体。代理位于客户端和服务器之间，充当“中间人”，在双方之间来回交换HTTP消息。

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服务器通信。使用HTTP代理，客户机转而与代理通信，代理本身代表客户机与服务器通信。

HTTP代理服务器既是web服务器又是web客户机。因为HTTP客户机向代理发送请求消息，所以代理服务器必须像web服务器一样正确处理请求和连接并返回响应。同时，代理本身将请求发送到服务器，因此它也必须像正确的HTTP客户机一样工作，发送请求和接收响应。

HTTP代理的工作模式如下图所示：



3 实验任务

3.1 实现你自己的HTTP服务器

在这个实验中，我们不提供任何基本代码。因此，您应该从头实现一个HTTP服务器(基于HTTP/1.1)，它满足以下要求:

3.1.1 HTTP服务器概要

从网络的角度来看，你的HTTP服务器应该实现以下功能:

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. 3.接受客户端并获得一个新的连接套接字
4. 读入并解析HTTP请求
5. 开始交付服务:
   1）处理HTTP GET/POST请求，如果发生错误，返回错误消息。
   2）代理请求到另一个HTTP服务器(进阶版本，可选)。

服务器将处于非代理模式或代理模式(我们在“2.3”节中介绍了代理)。它不会同时做这两件事。

3.1.2 处理HTTP请求

在这次实验中，你只需要在你的HTTP服务器中实现GET方法和POST方法。也就是说，如果你的HTTP服务器接收到既不是GET也不是POST的一个HTTP请求，只需要返回501未实现的错误消息（一条响应消息，其响应行中的状态码为501，参见`2.2`）

不需要处理请求中的标题行（但是您需要识别它，这样您就不会将它与下一个请求的起始行混合在一起）。另外，您可以随意填充HTTP响应中的任何(或零)标题行。

3.1.2.1 处理HTTP GET请求

HTTP服务器应该能够处理html页面的HTTP GET请求。

1. 如果HTTP请求的路径对应于一个html页面文件，则使用200 OK和该文件的完整内容进行响应。例如，如果请求GET /index.html，并且文件目录中存在一个名为index.html的文件。您还应该能够处理对文件目录(例如GET /images/hero.jpg)子目录中的文件的请求。
2. 如果HTTP请求的路径对应于一个目录，而该目录包含一个“索引”。以200 OK和该文件夹中index.html文件的全部内容作为响应。

3. 如果请求的页文件不存在，或请求的目录不包含`index.html`。返回一个404 Not Found响应(HTTP body是可选的)。

3.1.2.2 处理HTTP POST请求

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. 你应该构造一个包含2个key的HTTP POST请求(参见“3.1.7.2”小节)：“Name”和“ID”(请分别填写你的姓名和学号)，并将POST请求发送到`/Post_show`(即`http://127.0.0.1:8888/post_show`如果服务器的IP是`127.0.0.1`，服务端口是`8888`)。

然后，如果HTTP服务器接收到此POST请求(请求URL为`/Post_show`，keys为"Name"和"ID")，则使用200 OK进行响应，并回送您发送的"Name"-"ID"对(参见' 3.1.7.2 '小节)。

2. 否则(即请求URL不是`/Post_show`或key不是"Name"和"ID")，返回一个404 not Found响应消息。

3.1.2.3 其他请求

对于其他请求方法(如DELETE、PUT等，请参见“3.1.7.3”小节)，只返回501条未实现的错误消息。

3.1.3 实现一个代理服务器（可选）

允许您的服务器将HTTP请求代理到另一个HTTP服务器，并将响应转发给客户端。

1. 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. 您的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，您应该立即将其读入缓冲区，然后将其写入另一个套接字。您实际上是在维护HTTP客户机和上游HTTP服务器之间的双向通信。请注意，您的代理必须支持多个请求/响应。
3. 如果其中一个套接字关闭，通信将无法继续，因此应该关闭另一个套接字并退出子进程。

提示:

1）这比写入文件或从stdin中读取更棘手，因为您不知道双向流的哪一边将首先写入数据，或者它们是否在收到响应后将写入更多数据。

2）你应该再次使用线程来完成这个任务。例如，考虑使用两个线程来促进双向通信，一个是从A到B，另一个是从B到A。

3.1.4使用多线程来增加并发性

您的HTTP服务器应该使用多个线程来处理尽可能多的并发客户机请求。你至少有以下三个选项来架构你的多线程服务器:

- **随需应变的线程。**您可以在新客户端出现时创建一个新线程，并使用该线程处理所有客户端任务，包括解析HTTP请求、获取页面文件和发送响应。线程可以在客户端完成后销毁，例如，通过TCP `recv()`检测。然而，在HTTP层中检测客户端完成并不容易。

- **始终在线的线程池。**可以在HTTP服务器程序中使用固定大小的线程池来并发处理多个客户机请求。如果没有任务，则这些线程处于等待状态。如果有新客户机进来，分配一个线程来处理客户机的请求并向它发送响应。如果分配的线程很忙，您可以使用工作队列来缓冲请求，并让线程稍后处理它。
- **结合二者。**将以上两种风格结合在一起。例如，您可以使用线程池来接收请求和发送响应，并使用按需线程来获取大型页面文件。

您可以自由地从以上三种架构中选择任何一种，或者使用其他您认为很酷的多线程架构。

3.1.5 指定参数

您的程序应该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`,`--proxy`（可选）。

1. `--ip` ——指定服务器IP地址。
2. `--port` ——选择HTTP服务器侦听哪个端口的传入连接。
3. `--proxy` ——选择一个“上游”HTTP服务器作为代理。参数可以在冒号后面跟一个端口号(例如`https://www.cs06142.com: 80`)。如果没有指定端口号，则默认端口80。

如果你不知道*长选项*，你可以阅读this。您可能需要使用一些函数，如`getopt_long()`、`getopt_long_only()`、`getopt()`等等。使用`man`命令检查这些函数的用法。

3.1.6 运行你的HTTP服务器

你的程序应该可以在终端启动。如果你的程序名为*httpserver*，只需输入:

- 非代理模式:

```
./httpserver --ip 127.0.0.1 --port8888 --number-thread8
```

这意味着您的HTTP服务器的IP地址是127.0.0.1，服务端口是8888。`--number-thread`表示线程池中有8个线程用于并发地处理多个客户机请求。

- 在代理模式下:

```
./httpserver --ip 127.0.0.1 --port8888 --number-thread  --proxy https://www.cs06142.com:80
```

这意味着这是一个HTTP代理。这个代理的IP地址是127.0.0.1，服务端口是8888。代理有一个8个线程的线程池。`--proxy`表明“上游”HTTP服务器是`https://www.cs06142.com:80`。因此，如果您发送一个请求消息到这个代理(即。`127.0.0.1:8888`)，它会将此请求消息转发给"上游" HTTP服务器(即`https://www.cs06142.com:80`)，并将响应消息转发给客户端。

当您运行上面的命令时，您的HTTP服务器应该能够正确运行。

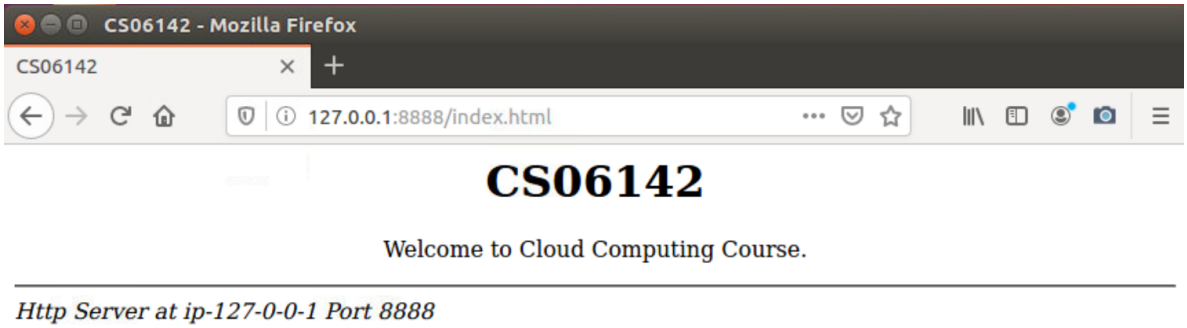
3.1.7 访问你的HTTP服务器

3.1.7.1 使用GET方法

1. 您可以通过打开web浏览器并转到适当的URL来检查HTTP服务器是否工作正常。

   [注]IP 127.0.0.1为本地主机的IP。因此，可以使用127.0.0.1在相同的本地机器上测试HTTP服务器。

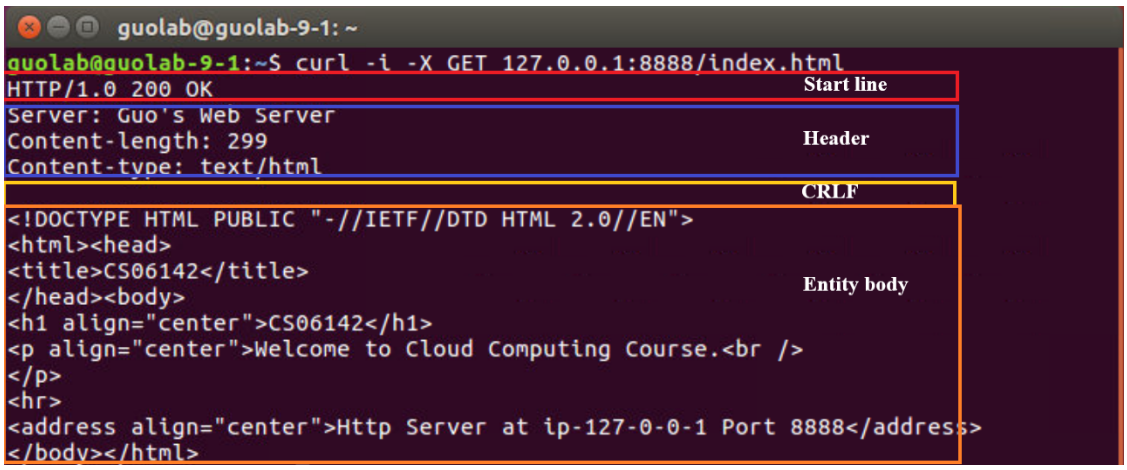
例如:



2. 您还可以使用curl程序发送HTTP请求。使用curl的一个例子:

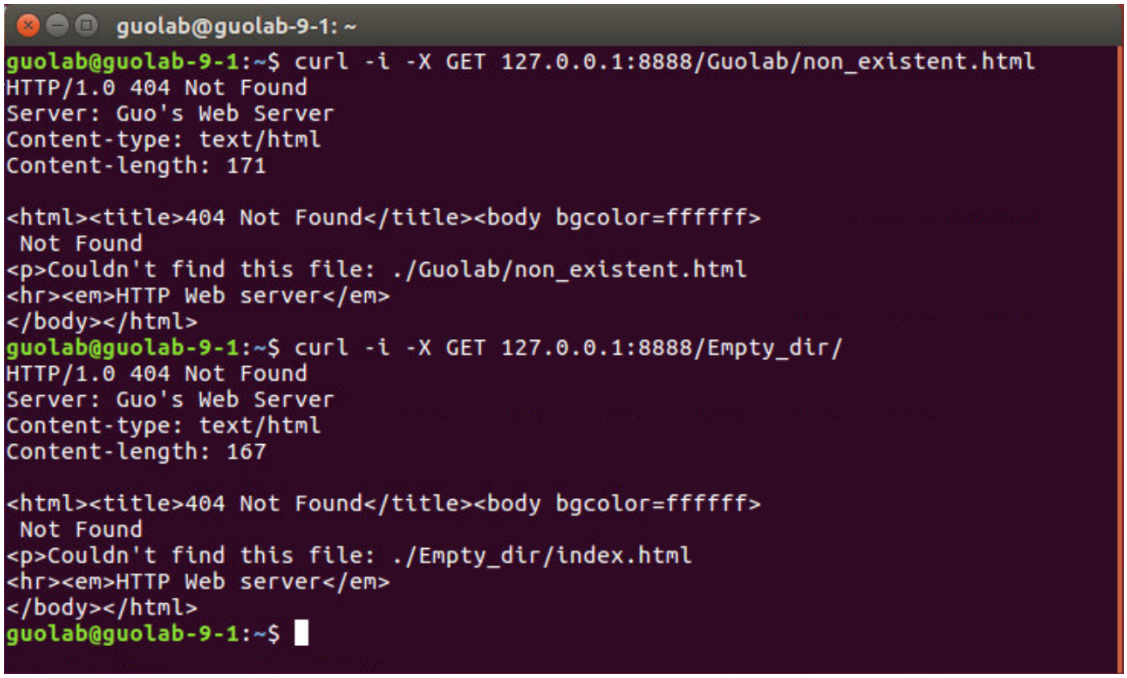
   `curl -i -X GET http://127.0.0.1:8888/index.html`

   例如:



3. 如果请求页面不存在，HTTP服务器应该返回404 Not Found错误消息。

   例如：

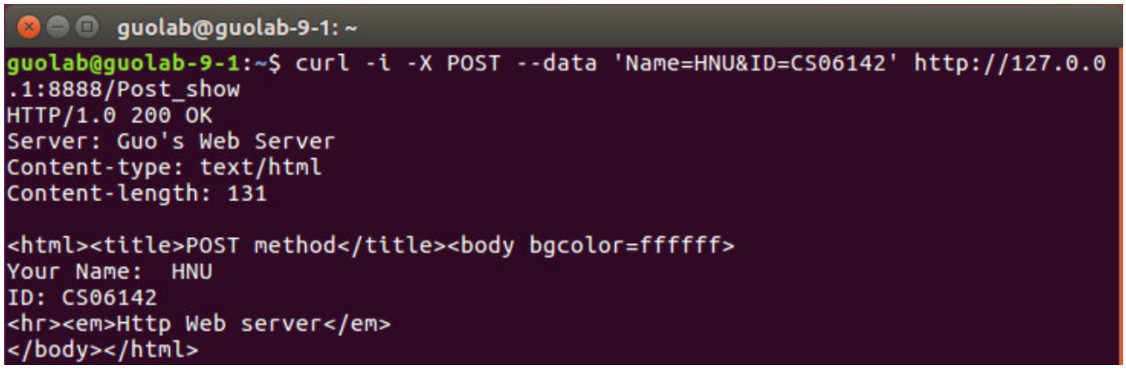


3.1.7.2 使用POST方法

1. 您可以通过curl程序发送一个HTTP请求来检查POST方法是否工作。在终端输入命令:

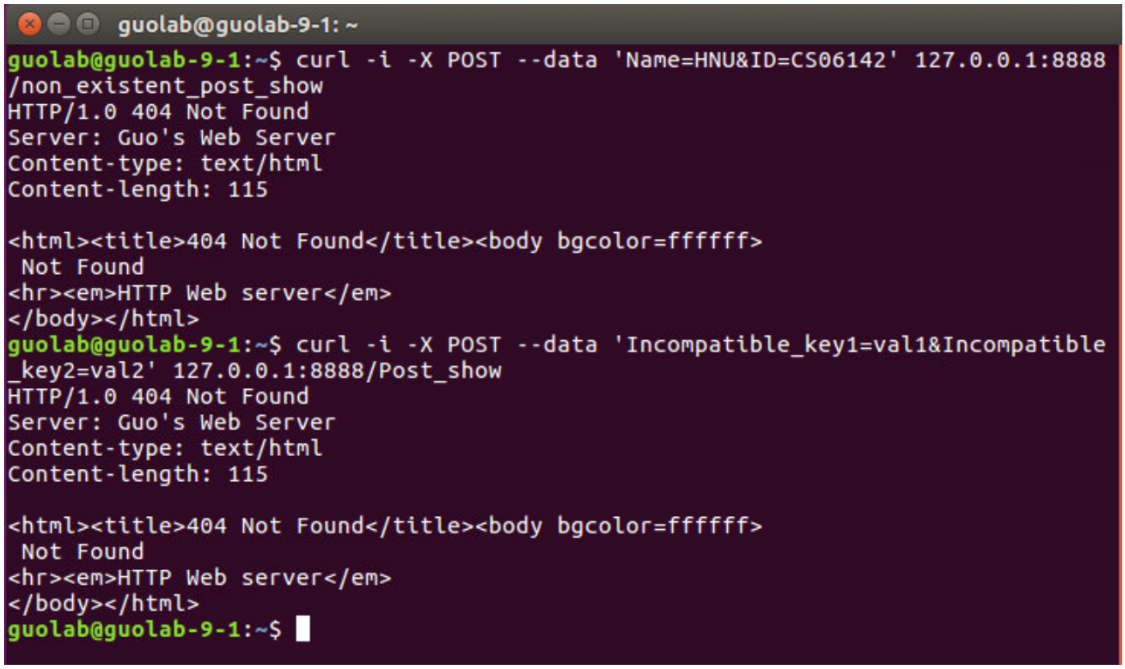
   `curl -i -X POST --data 'Name=HNU&ID=CS06142' http://127.0.0.1:8888/Post_show`

   例如：



2. 如果请求URL不是 `/Post_show` 或者键不是"Name"和"ID")，您将得到404 not Found错误消息。

   例如：

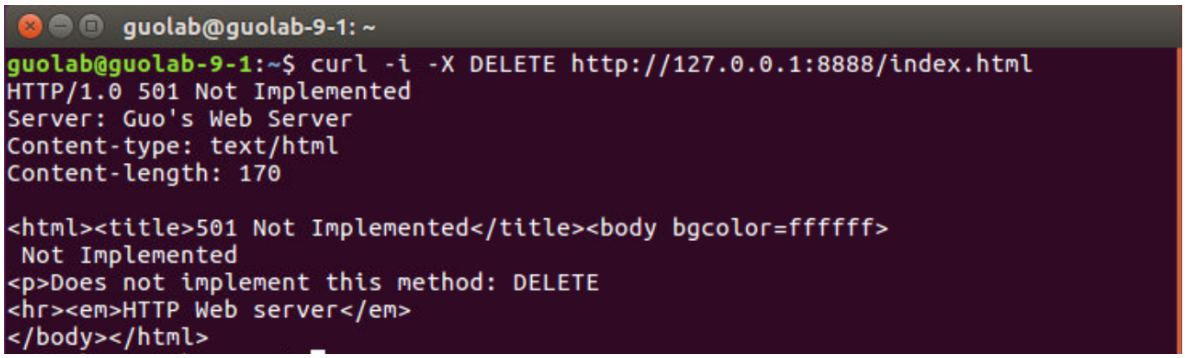


3. 您还可以构造POST HTTP请求，并使用一些浏览器插件工具将请求发送到HTTP服务器。

3.1.7.3 其他方法

除了GET请求和POST请求外，HTTP服务器将不处理HTTP请求。

如果你发送一个HTTP DELETE(或PUT, HEAD等)请求删除指定的资源，你将得到一个501未实现的错误消息:



3.1.8 实现要求

3.1.8.1 基础版本

您的程序应该完成 `3.1.1~3.1.7` 节中描述的所有任务，除了 `3.1.3`。

在基本版本中，每个TCP连接在同一时间只有一个请求。客户端等待响应，当它获得响应时，可能会为新请求重用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 进阶版本

您的程序应该完成 `3.1.1~3.1.7` 节中描述的所有任务，包括 `3.1.3`。

在高级版本中，可以在一个TCP连接上并发地发出多个http请求。这也被称为HTTP管道，它被许多实际的浏览器和服务器(如Chrome)所支持。注意，来自相同TCP连接的HTTP请求应该以相同的顺序响应。因此，在使用复杂的多线程样式时要注意顺序问题。

3.2 完成一个性能测试报告

请先测试您的代码，并将测试报告连同您的实验代码提交到您的小组课程github仓库中。

测试报告应描述在各种测试条件下的性能结果。具体来说，在你的测试报告中，你至少应该包含以下两点:

1. 测试服务器在各种服务器机器环境上运行时每秒可以处理多少HTTP请求。例如，改变服务器CPU内核的数量，启用/禁用超线程，等等。
2. 通过改变同时向服务器发送请求的并发客户机的数量，测试服务器每秒可以处理多少HTTP请求。请更改客户端的工作负载。例如，测试客户机何时为新请求使用新的TCP连接，或者客户机何时为新请求重用旧的TCP连接。此外，如果您实现了高级版本，请尝试更改同一客户端的TCP连接上的外边界请求数量。您可以编写一个简单的客户端,通过自己的发送HTTP Get（可以在同一台机器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客户端测试工具等(ab - Apache HTTP服务器基准测试工具) (http://httpd.apache.org/docs/current/programs/ab.html)。

[注]：注意客户端可能成为性能瓶颈。因此，在测试性能时最好使用多台机器。例如，您可以在三台机器(三个组成员)上运行多个客户机进程，在另一台机器(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可以达到性能限制。

4 实验提交

请将所有代码放在`Lab2`文件夹下，并写一个`Makefile`，让我们可以使用`make`编译你的代码。编译后的可执行文件要命名为`httpserver`，且也在`Lab2`下。

5 评分标准

1. 你可以得到23分如果你可以:
    - 完成基础版本的所有要求
    - 性能测试报告完成了上面说的两个要求
    - 如果你少了一些部分，将会根据具体完成多少得到部分的分数
2. 你可以得到25分如果你可以:
    - 完成进阶版本的所有要求
    - 性能测试报告完成了上面说的两个要求
    - 如果你少了一些部分，将会根据具体完成多少得到部分的分数