实验 2：您自己的 HTTP 服务器

一些材料来自 UC Berkeley 的 CS162 2019 的家庭作业 2。 感谢 CS162！ 输入您从我们的实验 git repo 克隆的文件夹，并提取最新的提交。

`git pull`

您可以在 `Lab2 / README.md` 中找到此 lab2 的指令。

Lab2 的所有材料都在文件夹 `Lab2 /`中

1. 概述

您可以使用从我们的课程中学到的网络编程知识，从头开始实施 HTTP 服务器。另外，请尝试使用从该类中学到的高并发编程技巧来保证 Web 服务器的性能。

实验目的：

- 练习基本的网络编程技能，例如使用套接字 API，解析数据包；
- 熟悉强大的高性能并发编程。

2. 背景

2.1 超文本传输协议

超文本传输协议（HTTP）是当今 Internet 上最常用的应用程序协议。 像许多网络协议一样，HTTP 使用客户端-服务器模型。 HTTP 客户端打开与 HTTP 服务器的网络连接，并发送 HTTP 请求消息。 然后，服务器以 HTTP 响应消息进行回复，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）。 为了方便起见，我们在本节中简要介绍 HTTP 消息格式和结构。 HTTP / 1.1 的详细规范可以在 RFC 2616 - Hypertext Transfer Protocol -- 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：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返回给客户端。请求和响应消息都具有相同的基本消息结构。

2.2.1 讯息格式

HTTP 请求和响应消息包含 3 个组件：

- 描述消息的开始行，
- 包含属性的标题块，
- 以及包含数据的可选正文。

每个组件的格式如下

2.2.1.1 起跑线

所有 HTTP 消息均以起始行开头。 请求消息的起始行说明了怎么做。 响应消息的起始行说明发生了什么。 具体来说，起始行在请求消息中也称为请求行，在响应消息中也称为响应行

1. **请求行：**请求行包含描述服务器应执行的操作的方法和描述执行该方法的资源的

请求 URL。 请求行还包含一个 HTTP 版本，该版本告诉服务器客户端使用哪种 HTTP 方言。 所有这些字段都由空格分隔。

请求行示例：

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

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

响应行示例：

```
HTTP/1.0 200 OK
```

2.2.1.2 标头

在开始行之后是一个零，一个或多个 HTTP 标头字段的列表。 HTTP 标头字段为请求和响应消息添加了其他信息。 它们基本上只是名称/值对的列表。 每个 HTTP 标头都有一个简单的语法：名称，后跟冒号（:），后跟可选的空格，后跟字段值，然后是 CRLF。 HTTP 标头分为：通用标头，请求标头，响应标头，实体标头和扩展标头。 请注意，请求标头字段与响应标头字段不同。 我们不会详细介绍这些领域，并且您无需在本实验中实施。 您可以在 RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1 中找到它们。

请求中的标头示例：

```
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
```

响应中的标头示例：

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

2.2.1.3 实体主体

HTTP 消息的第三部分是可选的实体主体。 实体是 HTTP 消息的有效负载。 它们是 HTTP 旨在传递的东西。 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，电子邮件等等。

实体的示例：

```
<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>
```

2.2.2 HTTP 请求的结构

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 标头行和空白行（即 CRLF）。

HTTP 请求消息的示例：

```
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 HTTP 响应的结构

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 标头行，空行（即，CRLF 本身）和内容 HTTP 请求所请求。

HTTP 响应消息示例：

```
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 代理

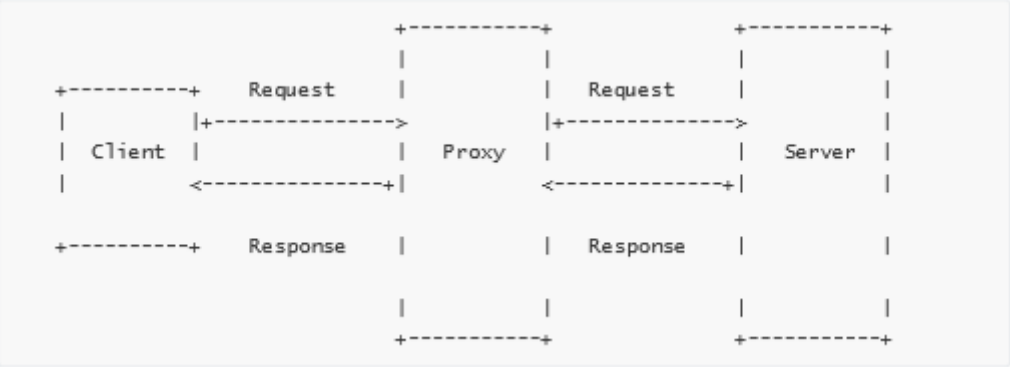
HTTP 代理服务器是中介。 代理位于客户端和服务器之间，并充当“中间人”，在双方之

间来回发送 HTTP 消息。

HTTP 代理服务器是代表客户端完成事务的中间商。 如果没有 HTTP 代理，HTTP 客户端将直接与 HTTP 服务器通信。 客户端使用 HTTP 代理与代理进行对话，而代理本身又代表客户端与服务器通信。

HTTP 代理服务器既是 Web 服务器又是 Web 客户端。 因为 HTTP 客户端将请求消息发送到代理，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 客户端一样工作，发送请求并接收响应。

HTTP 代理的工作模式如下图所示：



3. 您的实验任务

3.1 实施自己的 HTTP 服务器

在本实验中，我们将不提供任何基本代码。 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 服务器概述

从网络角度来看，您的 HTTP 服务器应实现以下功能：

1. 创建一个侦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. 读取并解析 HTTP 请求
5. 开始提供服务：
    1). 处理 HTTP GET / POST 请求，如果发生错误，则返回错误消息。
    2). 将请求代理到另一个 HTTP 服务器（高级版本可选）。

服务器将处于非代理模式或代理模式（我们已在 2.3 节中介绍了代理）。 它不会同时做两件事。

3.1.2 处理 HTTP 请求

在本实验中，**您只需要在 HTTP 服务器中实现 GET 方法和 POST 方法**。 也就是说，如果您的 HTTP 服务器接收到 HTTP 请求，但是请求方法既不是 GET 也不是 POST。 HTTP 服务器仅需要返回 501 Not Implemented 错误消息（响应消息的响应行的状态码为 501，请参见 2.2）。

不需要处理请求中的标题行（但是您需要识别它，以便您不会将其与下一个请求的起始行混合使用）。另外，您可以随意在 HTTP 响应中填充任何（或零个）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 请求的路径与 html 页面文件相对应，则以 200 OK 和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。 您还应该能够处理文件目录的子目录中的文件请求（例如 GET /images/hero.jpg）。
2. 如果 HTTP 请求的路径与目录相对应，并且该目录包含 index.html 文件，请在该文件夹中以 200 OK 以及 index.html 文件的完整内容进行响应。
3. 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）。

请参阅第 3.1.7.1 节中的示例。

3.1.2.2 处理 HTTP POST 请求

HTTP 服务器应该能够处理 HTTP POST 请求。 在本实验中，处理 HTTP POST 的方法非常简单。

1. 您应该构造一个包含 2 个键的 HTTP POST 请求（请参阅第 3.1.7.2 节）：“名称”和“ ID”（请分别填写您的姓名和学生编号），然后将 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 并且服务端口是 8888）。

然后，如果 HTTP 服务器接收到此 POST 请求（请求 URL 为/ Post_show，并且键为“ Name”和“ ID”），则响应为 200 OK，并回显您已发送的“ Name”-“ ID”对。

2. 否则（例如，请求 URL 不是/ Post_show 或键不是“ Name”和“ ID”），请返回 404 Not Found 响应消息。

请参阅第 3.1.7.2 节中的示例。

3.1.2.3 其他要求

只需为其他请求方法（例如 DELETE，PUT 等）返回 501 Not Implemented 错误消息。 请参阅第 3.1.7.3 节中的示例。

3.1.3 实施代理服务器（可选）

使您的服务器能够将 HTTP 请求代理到另一个 HTTP 服务器，并将响应转发给客户端。

1. 您应该使用--proxy 命令行参数的值，该值包含上游 HTTP 服务器的地址和端口号。
2. 您的代理服务器应在两个套接字（HTTP 客户端 fd 和上游 HTTP 服务器 fd）上等待新数据。数据到达后，您应该立即将其读取给购买者然后将其写入另一个套接字。本质上，您正在维护 HTTP 客户端和上游 HTTP 服务器之间的双向通信。请注意，您的代理服务器必须支持多个请求/响应。
3. 如果其中一个套接字关闭，则通信无法继续，因此您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 读取要棘手，因为您不知道 2 路流的哪一侧将首先写入数据，或者不知道在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，一个线程从 A 到 B，另一个线程从 B 到 A。

3.1.4 使用多线程来提高并发性

您的 HTTP 服务器应使用多个线程来处理尽可能多的并发客户端的请求。您至少具有以下三个选项来构建多线程服务器：

- 按需线程。每当有新客户端进入时，您都可以创建一个新线程，并使用该线程来处理所有客户端的任务，包括解析 HTTP 请求，获取页面文件和发送响应。客户端完成后，例如通过 TCP recv（）检测，可以销毁线程。但是，在 HTTP 层中检测客户端完成可能并不容易。
- 永远在线的线程池。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。如果有新客户端进入，请分配一个线程来处理该客户端的请求并发送响应。如果分配的线程繁忙，则可以使用工作队列为请求添加缓冲区，然后让线程稍后对其进行处理。
- 合并。将以上两种样式结合在一起。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页面文件。

随意选择以上三个中的任何一个，或使用您认为很酷的其他多线程体系结构。

3.1.5 指定参数

您的程序应启用长选项以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 -指定服务器 IP 地址。
2. –port -选择 HTTP 服务器侦听传入连接的端口。
3. –proxy -选择一个“上游” HTTP 服务器进行代理。 该参数可以在冒号后面有一个端口号（例如 https://www.CS06142.com:80）。

如果未指定端口号，则默认为端口 80。 如果您不知道长选项，可以阅读此内容。 并且您可能需要使用一些函数，例如 getopt_long（），getopt_long_only（）和 getopt（）等。 使用 man 命令检查这些功能的用法。

3.1.6 运行您的 HTTP 服务器

您的程序应该能够在终端启动。如果您的程序称为 httpserver，只需在非代理模式下键入：

```
./httpserver --ip 127.0.0.1 --port 8888 --number-thread 8
```

这意味着您的 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 个线程可同时处理多个客户端请求。

在代理模式下：

```
./httpserver --ip 127.0.0.1 --port 8888 --number-thread 8 –proxy
https://www.CS06142.com:80
```

这意味着这是一个 HTTP 代理。该代理的 IP 地址为 127.0.0.1，服务端口为 8888。该代理具有 8 个线程的线程池。 --proxy 表示“上游” HTTP 服务器为 https://www.CS06142.COM:80。因此，如果您向该代理发送请求消息（即 127.0.0.1:8888），它将将该请求消息转发至“上游” HTTP 服务器（即 https://www.CS06142.com:80）并转发给客户端的响应消息。

当您运行上面的命令时，您的 HTTP 服务器应该正确运行。

3.1.7 访问您的 HTTP 服务器

3.1.7.1 使用 GET 方法

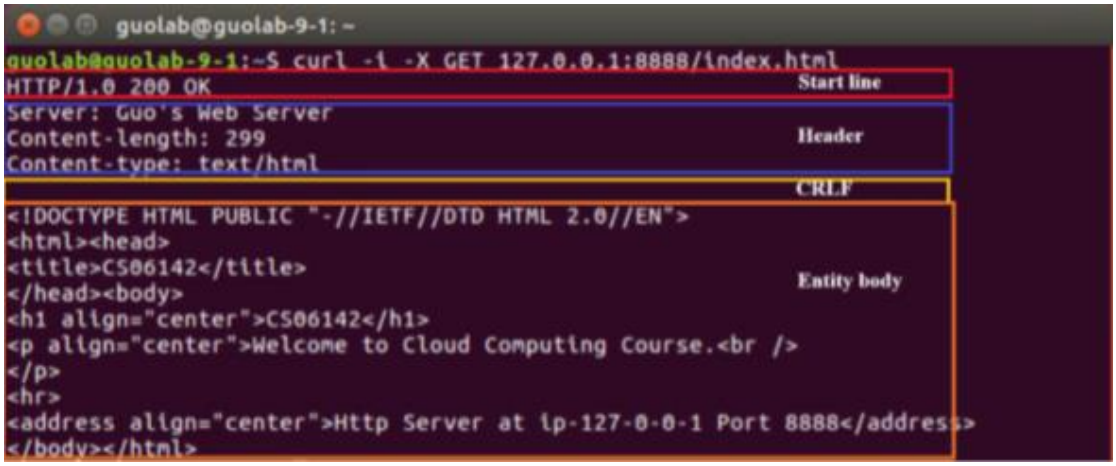
1. 您可以通过打开 Web 浏览器并转到相应的 URL 来检查 HTTP 服务器是否正常工作。 [注] IP 127.0.0.1 是指本地主机的 IP。 因此，您可以使用 127.0.0.1 在同一台本地计算机上测试您的 HTTP 服务器。

例如：

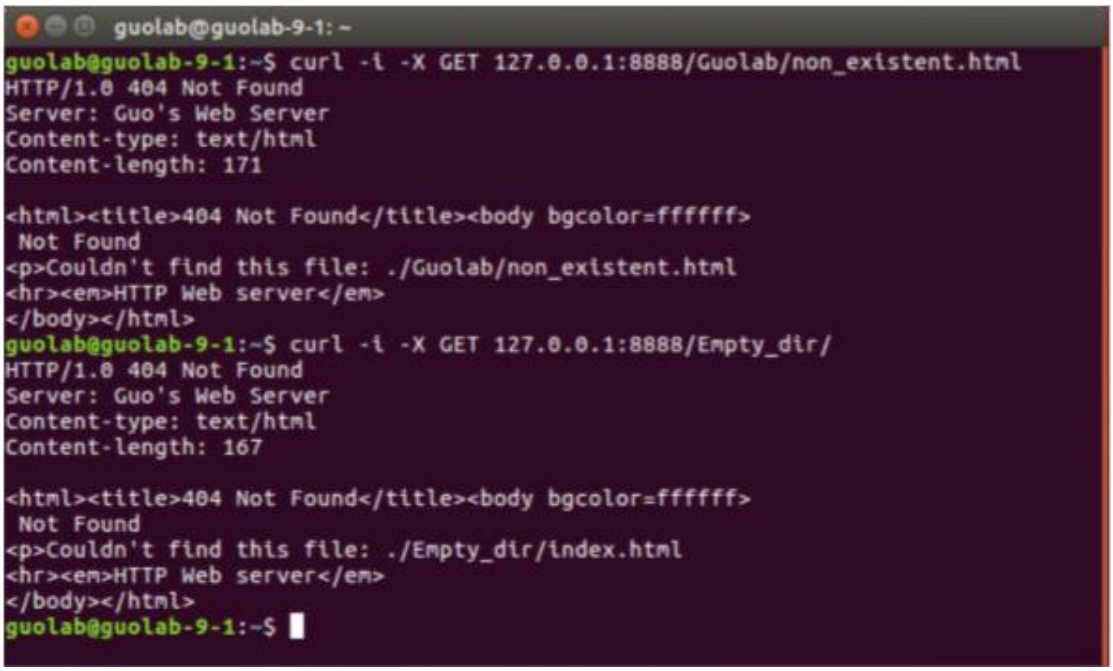
您还可以使用 curl 程序发送 HTTP 请求。 如何使用 curl 的示例是：

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

例如



2. 如果请求页面不存在，则您的 HTTP 服务器应返回 404 Not Found 错误消息。
例如：

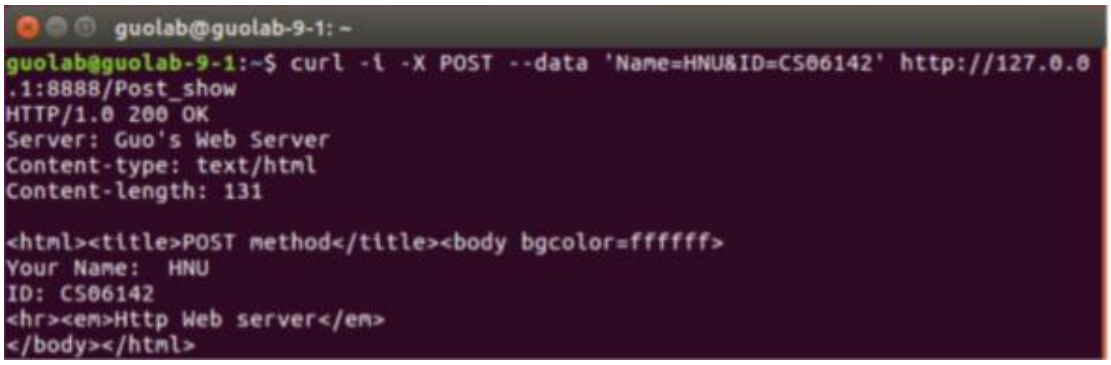


3.1.7.2 使用 POST 方法

1. 您可以通过使用 curl 程序发送 HTTP 请求来检查 POST 方法是否有效。 在终端上键入命令：

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

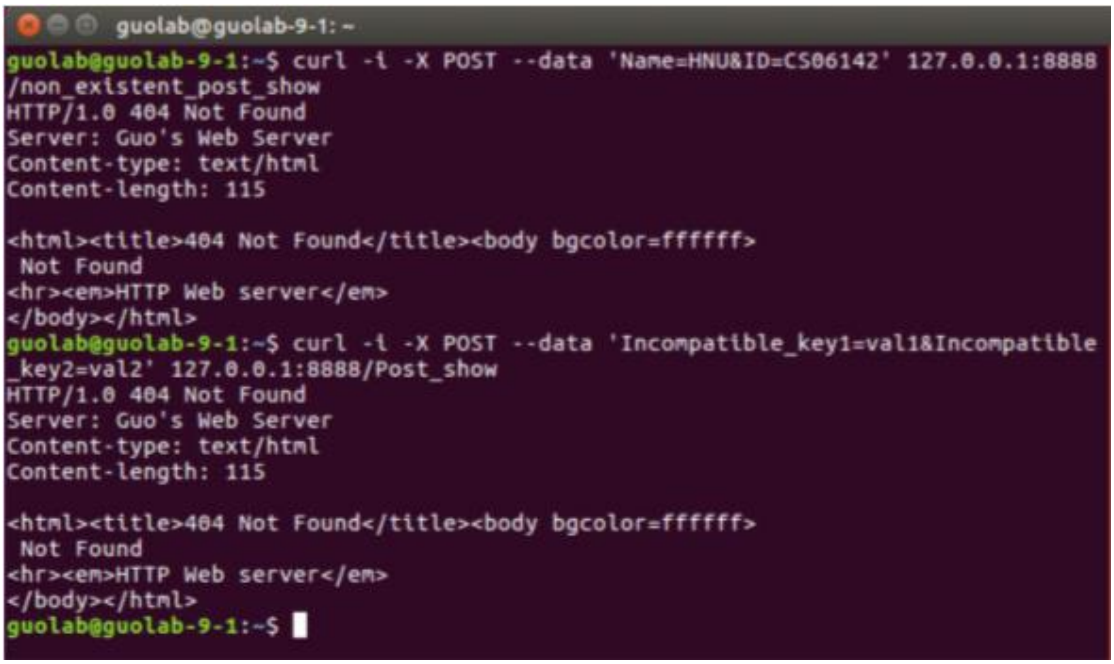
例如：



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

2. 你好如果请求 URL 不是/ Post_show 或键不是“ Name”和“ ID”），则会收到 404 Not Found 错误消息。

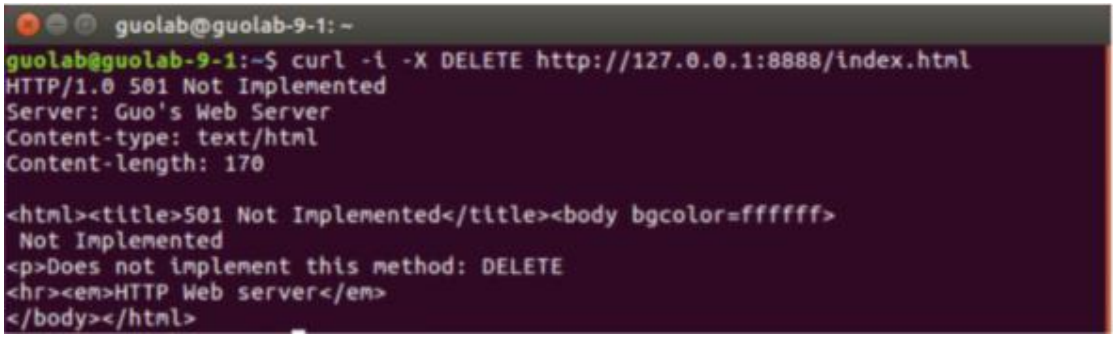
例如：



3.1.7.3 其他方法

除 GET 请求和 POST 请求外，HTTP 服务器将不处理 HTTP 请求。

如果您发送 HTTP DELETE（或 PUT，HEAD 等）请求以删除指定的资源，则会收到 501 Not Implemented 错误消息：



3.1.8 实施要求

3.1.8.1 基本版本

您的程序应完成 3.1.1~3.1.7 中描述的所有任务，但 3.1.3 除外。
在基本版本中，**每个 TCP 连接只能同时发送一个请求**。 客户端等待响应，当客户端获得响应时，也许将 TCP 连接重新用于新请求（或使用新的 TCP 连接）。 这也是普通 HTTP 服务器支持的内容。

3.1.8.2 进阶版

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。
在高级版本中，**可以在一个 TCP 连接上同时发出多个 http 请求**。 这也称为 HTTP 管道传输，许多实际的浏览器和服务器（例如 Chrome）都支持 HTTP 管道传输。 请注意，来自同一 TCP 连接的 HTTP 请求应以相同的顺序响应。 因此，在使用复杂的多线程样式时请注意顺序问题。

3.2 完成性能测试报告

请首先测试您的代码，然后将测试报告以及您的实验室代码提交到小组的课程 github 存储库中。
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，您至少应包含以下两件事:

1. 在各种服务器计算机环境上运行时，测试您的服务器每秒可以处理多少个 HTTP 请求。例如，更改服务器 CPU 内核数，启用/禁用超线程等。
2. 通过更改同时发送请求到服务器的并发客户端数，测试服务器每秒可以处理多少个 HTTP 请求。一定要改变客户的工作量。例如，测试客户端何时将新的 TCP 连接用于新请求，或何时客户端将旧的 TCP 连接重新用于新请求。此外，如果实施高级版本，请尝试更改同一客户端的 TCP 连接上的出站请求数。您可以编写一个自己发送 HTTP Get 的简单客户端（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），也可以使用一些现有的 HTTP 客户端测试工具，例如 ab - Apache HTTP server benchmarking tool。

[注意]：请注意，客户可能是性能瓶颈。因此，在测试性能时，最好使用多台计算机。例如，您可以在（三个组成员的）三台计算机上运行多个客户端进程，并在（另一个组成员的）另一台计算机上运行服务器进程。而且，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极限，并查看您的实现是否可以达到性能极限。

4. 实验室提交

请将所有代码放在 Lab2 文件夹中并编写一个 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，位于文件夹 Lab2 中。 请仔细遵循上述规则，以便 TA 可以自动测试您的代码!!! 请按照 Overall Lab Instructions（../README.md）中的指导提交实验方案和性能测试报告。

5. 分级标准

1. 如果可以，您将获得 23 分：
   1). 完成基本版本的所有要求，以及
   2). 性能测试报告已满足上述两个要求。 如果您错过了某些部分，您将获得部分积分，具体取决于完成的数量。

2. 如果可以，您将获得 25 分（满分）：
   1). 完成高级版本的所有要求，以及
   2). 性能测试报告已经满足了上述两个要求。 如果您错过了某些部分，您将获得部分积分，具体取决于完成的数量。