天津大学

《计算机网络》课程设计
周进度报告

第二周 实现 HEAD、GET 和 POST 方法

学　　号 3020205015 3020210104

姓　　名 石云天　姜卓雨

学　　院 未来技术学院

专　　业 计算机科学与技术

年　　级 2020

任课教师 周晓波

2023 年 6 月 8 日

目　录

一、协议设计

1.1 协议头部

在 HTTP 协议中，请求报文的协议头部由请求行(request line)和首部行(header line)组成，其中请求行记录了方法、URL、版本。响应报文的协议头部由状态行(status line)和首部行（header line）组成，其中状态行记录了版本、状态码、短语。

（1）方法：本次实验需要实现的方法为 GET、HEAD 和 POST 方法。

（2）URL：记录了请求的地址对象。

（3）版本：本次实验需要实现的版本为 HTTP/1.1。

（4）状态码：记录响应的报文的代码状态，常用的有 200、301、400、403、404、501、505 等。

（5）短语：状态码对应的状态短语。如 200 对应的“OK”，400 对应的“Bad Request”，404 对应的“Not Found”，501 对应的“Not Implemented”等。

1.2 协议规则

（1）GET

请求指定的页面信息并返回实体主体。本质是发送一个请求来取得服务器上的某一资源。资源通过一组 HTTP 头和呈现数据（如 HTML 文本，或者图片或者视频等）返回给客户端。在 GET 请求中，不会包含呈现数据。

特性：只允许 ASCII 字符数据类型，能被缓存，但发送的数据是 URL 的一部分，因此安全性较差。

（2）HEAD

向服务器索与 GET 请求相一致的响应，但响应体将不会被返回。可在不必传输整个响应内容的情况下，就获取包含在响应小消息头中的元信息。

（3）POST

向指定资源提交数据进行处理请求（例如提交表单或者上传文件)。数据被包含在请求体中。POST 请求可能会导致新的资源的建立或已有资源的修改。

特性:不能缓存，对数据类型与数据长度没有限制(允许二进制)，参数不会保存在浏览器历史或 web 服务器日志中，更为安全。

1.3 数据结构与算法

本次实验中缓冲区的数据结构为顺序结构，采用数组存储。在实验中，服务端程序完成初始化后，首先和客户端建立连接，然后将接收到的数据存入缓冲区，通过语法分析器分析缓冲区中的请求报文，根据分析结果确定响应报文格式，将响应报文写入缓冲区并发送给客户端，完成后关闭连接。

接收缓冲区采用限制最大读取字节数的方式保证缓冲区不会溢出。

日志记录模块仿照 Apache HTTP 服务器 2.4 实现，通过调用 write_access_log 和 write_error_log 函数来实现对日志的记录。

二、协议实现

2.1 响应 POST 方法及未实现方法

由于在第一阶段已完成对请求报文的解析和返回，因此本次实现 POST 方法及未实现方法只需在第一阶段的基础上增加判断 request 方法是否为 POST 方法，如果是 POST 方法，则直接将接收到的报文 echo 回去。如果是未实现方法，则返回“HTTP/1.1 501 Not Implemented\r\n\r\n”。

除此之外，本次实验还要求对 HTTP 版本进行校验，如果是未实现的 HTTP 版本(如 HTTP/1.0)，则返回“HTTP/1.1 505 HTTP Version Not Supported\r\n\r\n ”。

2.2 完善 echo_server

由于 GET、HEAD 方法需要查找 URL 对应的文件，所以在实现 GET、HEAD 方法前需要对 URL 进行解析。由于本地存储静态网站的位置在 static_site 文件夹下，所以需要在请求的 URL 前加上 static_site。由于在请求报文中传递的 URL 的格式为 abs_path[""query]，而服务器并未实现对查询的响应，因此需要去除 URL 中的“？”及之后的所有查询。除此之外，为了实现对特殊字符甚至是中文字符的解析，服务端需要将“%”及其之后的两个字符转换为一个十六进制数对应的字符(需要解析使用 UTF-8 的 HEX 编码)，如字符串“%E4%B8%AD%E6%96%87”解析之后需要得到的字符串为“中文”。

为了实现这些需求，需要对 URL 逐个字符进行扫描。当扫描到“+”时，将其替换为“ ”(空格在 URL 中需要转换为“+”或“%20”，否则服务器将无法解析请求报文，而在解析时需要将"+"或"%20"替换回空格)；当扫描到“%”时，需要将其以及其后 2 个字符替换为其对应的字符，如“%E4”需要替换为(char)-28

（负数的字符型数据通常是多字节字的其中一个字节对应的字符)；当扫描到“?”时，需要忽视其自身以及其之后的所有字符，即将“？”替换为“\0”。经过上述解析后，便可通过该 URL 指定的相对地址访问资源。

2.3 响应 HEAD 方法

在解析完 URL 后，需要查看 URL 指定的文件是否存在，因此首先使用 stat 函数判断 URL 指定的位置是否存在，如果不存在则返回“HTTP/1.1 404 NotFoundVrinirin”。如果存在，则判断该地址是否为文件夹，如果是文件夹则在 URL 后添加“index.html”作为默认读取的文件。然后根据新的地址重新查看文件是否存在，如果不存在则返回“HTTP/1.1 404 Not Foundlrlntrln”。如果存在，则判断该文件是否可读取，如果不可读取，则返回"HTTP/1.1 403 Forbiddenkrinirln"。如果可读，则判断文件的媒体类型，如“text/htm1”、“text/css”、“image/png”等，将得到的媒体类型字符串写入 filetype 备用。

接下来，将响应报文的协议头部写入缓冲区。首先是 status line，由于此时已确认文件可以访问，因此 status line 应为“HTTP/1.1 200 OK\rln”。然后是 headerline，由于 HTTP 1.1 默认为持久性连接，因此将“Connection: keep-alivelrln”写入缓冲区;由于本次实验实现的服务器名为 liso，所以将“Server: liso/1.0\rln”写入缓冲区;然后按照 RFC2616 的规定，在 header line 中写入“Date”行，即当前时间;之后是文件信息，需要在 header line 中写入的有“Content-Type”行、“Content-Length”行、“Last-Modified”行，其中“Content-Type”行的信息由上文中的 filetype 指定，而“Content-Length”行和“Last-Modified”行中的信息需要从之前用 stat 函数访问文件得到的结果中获取。之后在缓冲区中写入“\r\n”表示响应报文的协议头部结束，之后将缓冲区中的内容发送回客户端即可实现 HEAD 方法的响应。

2.4 响应 GET 方法

GET 方法在 HEAD 方法的基础上，还需要返回响应体，即请求的文件内容。因此使用 fopen 函数读取 URL 指定的文件。首先检查文件内容是否能够在一次内发送完成(本次实验设定缓冲区大小为 8192 字节)。如果可以，则将响应头部连带整个文件一次发送出去;如果不能，则从文件中读取缓冲区剩余空间大小的数据，附在响应头部之后并发送，之后每次都从文件中读取缓冲区大小的数据并发送直到文件发送完成。此时客户端会根据 header line 中“Content-Length”行指定的响应体大小读取数据，因此多次发送响应报文并不会出现问题。

2.5 创建日志

（1）错误日志 error_log

在创建错误日志之前，首先仿照 Apache HTTP 服务器对日志记录等级进行定义。在 Apache HTTP 服务器 2.4 中，定义了 16 种日志记录等级，按照严重性递减的顺序分别为"emerg" "alert""crit""error" "warn""notice""info" "debug""trace1" "trace2""trace3" "trace4" "trace5" "trace6" "trace7""trace8"。我们将"emerg""alert" "crit"合并为"fatal"，将"notice""info"合并为"info"，将"tracel"~"trace8"合并为"trace"，构成了"fatal""error" "warn""info" "debug""trace"6 个日志记录等级，分别赋值为 5~0，作为日志记录等级编号。

Apache HTTP 服务器 2.4 对于问题日志的默认记录格式为 ErrorLogFormat"[%t] [%l1] [pid %P]%F: %E:[client %a] %M"，典型的日志信息如"[Fri Sep 0910:42:29.902022 2011] [core:error] [pid 35708:tid 4328636416] [client 72.15.99.187]File does not exist: /usr/local/apache2/htdocs/favicon.ico"所示。仿照 Apache 日志的记录方式，我们自行定义了一个简易的错误日志，格式为"[%t] [%l] [pid %P] [client %a] %m: %M"(其中%t 为当前时间，%1 为消息的日志级别，%P 为当前进程的进程 ID，%a 为客户端 IP 地址和请求的端口，%m 为记录消息的模块名称，%M 为实际的日志消息)，典型的日志信息如"[Tue Nov 29 03:23:06 2022] [error] [lpid 19473] [client 127.0.0.1:49064] socket: Connection reset by peer"所示。

为了方便错误日志的记录，错误日志记录封装为 write_error_log 函数，参数有 level-问题等级、addr-来源地址、src-来源、msg-问题详情。需要记录错误日志时，直接调用该函数即可。

（2）访问日志 access_log

Apache HTTP 服务器 2.4 对于问题日志的默认记录格式为 LogFormat"%h %1l %u %t \"%r"”%>s %b" common，典型的日志信息如"127.0.0.1 - frank[10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gifHTTP/1.0" 2002326"所示。对于访问日志，在发送响应报文后，通过调用 write_access_log 函数，来记录访问信息。write_access_log 函数的参数有 addr-访问者信息、code 状态代码、sent 已发送字节数。

2.6 其他问题

（1）缓冲区溢出问题的处理

为了防止缓冲区溢出，在接收数据和发送数据时都会对数据大小进行限制。在本次实验中，我们设置缓冲区的大小为 8192 字节。每次读取数据时，都从 socket 的接收缓冲区读取最多 8192 字节的数据到用户缓冲区。此时，如果请求头部过

长，则会使得缓冲区无法完整保存请求头部，从而导致语法分析失败。因此，当请求头部大于 8192 字节时，会返回"HTTP/1.1 400 Bad Request\r\n\r\n "。而每次发送数据时，都会保证响应头部不超过 8192 字节，而响应体如果过大，无法在一个数据包内发送完毕，则会将响应体分割，多次发送。由此，便可保证缓冲区不会溢出。

（2）读写磁盘遇到错误的处理

在本实验中，涉及到写磁盘的操作只有写日志。如果刚开始打开日志文件出现错误时，则报错并退出程序。而涉及到读磁盘的操作有根据 URL 读文件，如果通过 stat 函数已经确认文件可以读取，但在实际读取时，由于种种原因文件又不能读取时，会向客户端返回"HTTP/1.1 500 Internal Server Error\r\n\r\n"，不会影响服务器的正常运行。

三、实验结果及分析

3.1 GET 功能测试

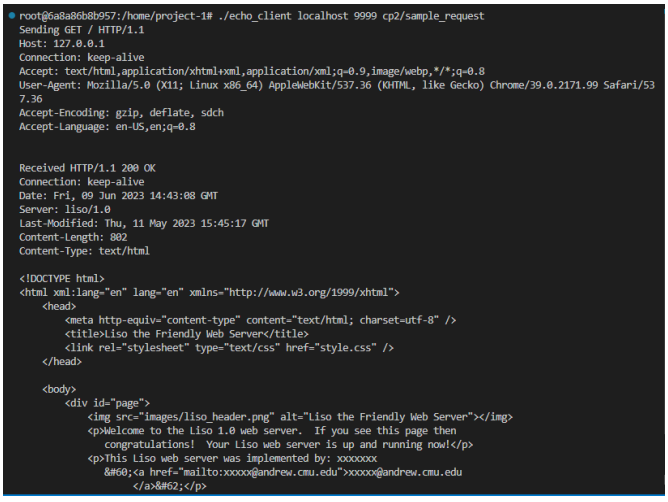

图 1 GET 测试结果

发送报文：

```
./echo_client localhost 9999 cp2/sample_request
Sending GET / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
```

```
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

接收报文：

```
Received HTTP/1.1 200 OK
Connection: keep-alive
Date: Fri, 09 Jun 2023 14:43:08 GMT
Server: liso/1.0
Last-Modified: Thu, 11 May 2023 15:45:17 GMT
Content-Length: 802
Content-Type: text/html
<!DOCTYPE html>
<html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-
8" />
        <title>Liso the Friendly Web Server</title>
        <link rel="stylesheet" type="text/css" href="style.css" />
    </head>
    <body>
        <div id="page">
            <img src="images/liso_header.png" alt="Liso the Friendly Web
Server"></img>
            <p>Welcome to the Liso 1.0 web server.  If you see this page
then
               congratulations!  Your Liso web server is up and running
now!</p>
            <p>This Liso web server was implemented by: xxxxxxx
               &#60;<a
href="mailto:xxxxx@andrew.cmu.edu">xxxxx@andrew.cmu.edu
                    </a>&#62;</p>
        </div>
    </body>
</html>
```

3.2 HEAD 功能测试

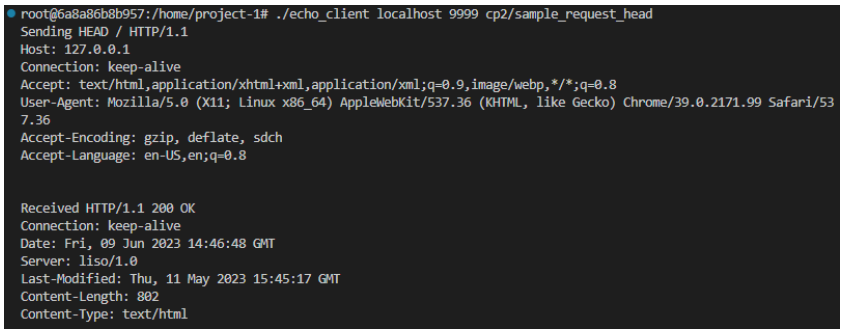

图 2 HEAD 测试结果

发送报文：

```
Sending HEAD / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

接收报文：

```
Received HTTP/1.1 200 OK
Connection: keep-alive
Date: Fri, 09 Jun 2023 14:46:48 GMT
Server: liso/1.0
Last-Modified: Thu, 11 May 2023 15:45:17 GMT
Content-Length: 802
Content-Type: text/html
```

3.3 POST 功能测试

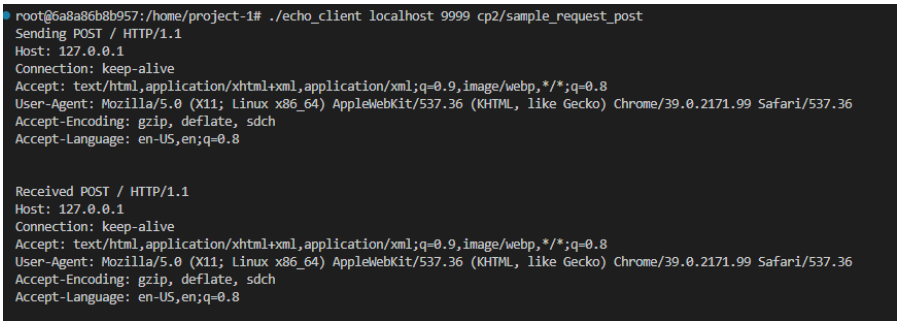

图 3 POST 功能测试

发送报文：

```
Sending POST / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

接收报文：

```
Received POST / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

3.4 测试其他方法（以 put 为例）

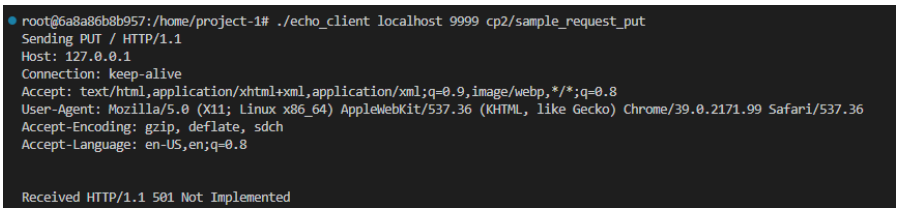

图 4 测试结果

发送报文：

```
Sending PUT / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

接收报文：

```
Received HTTP/1.1 501 Not Implemented
```

3.5 测试不同 HTTP 版本识别

图 5 测试结果

发送报文：

```
Sending GET / HTTP/1.0
Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
```

User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

接收报文：

Received HTTP/1.1 505 HTTP Version not supported

3.6 测试未找到文件

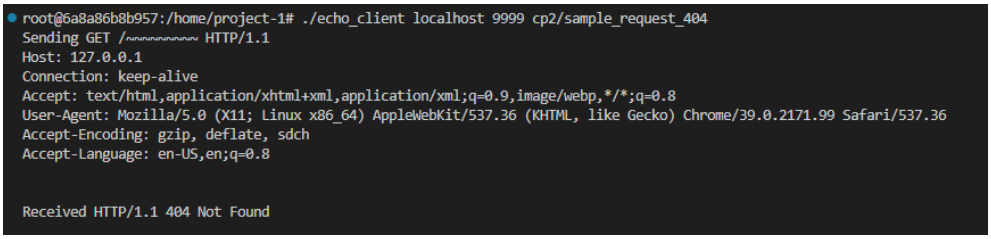

图 6 测试结果

发送报文：

Sending GET /~~~~~~~~~~ HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

接收报文：

Received HTTP/1.1 404 Not Found

3.7 测试错误报文段

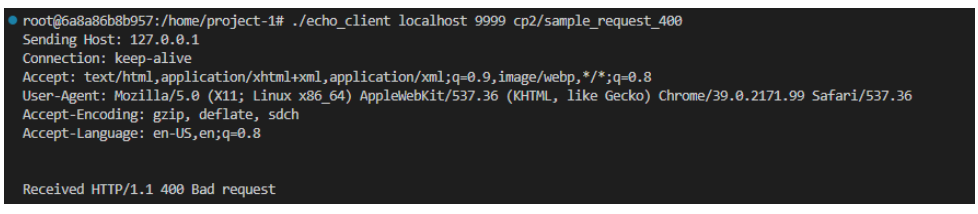

图 7 测试结果

发送报文：

```
Sending Host: 127.0.0.1
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

接收报文：

```
Received HTTP/1.1 400 Bad request
```

3.8 日志功能测试

日志能记录正常发送信息：

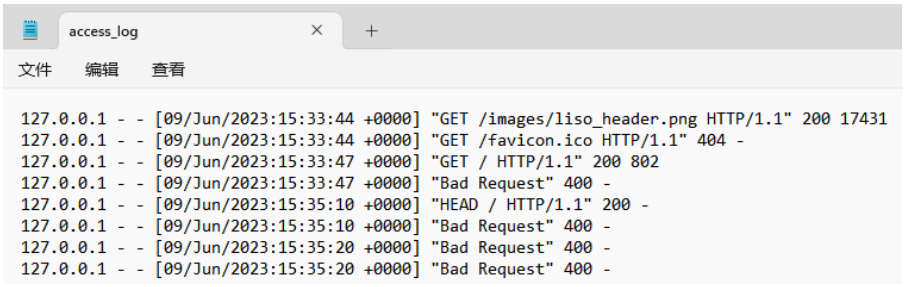

图 8 日志信息

日志能记录正常记录报错信息：

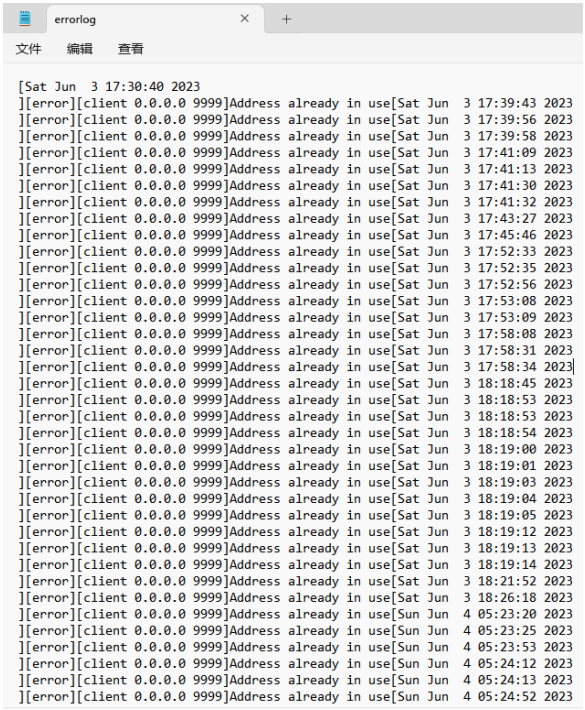

图 9 报错信息

3.9 浏览器测试

图 10 测试结果

四、进度总结

表 1 本周任务完成表

本阶段任务要求	完成	未完成	备注
1.1 响应 HEAD 方法	√		
1.2 响应 GET 方法	√		
1.3 响应 POST 方法	√		
1.4 支持 4 种 HTTP 出错代码	√		
1.5 妥善管理接收缓冲区	√		
2 服务器能够处理读写磁盘文件时遇到的错误	√		
3 创建简化的日志记录模块	√		
4. 功能测试	√		

表 2 上周任务改进表

序号	上周任务	改进内容	备注
1	无		