

天津大学

《计算机网络》课程设计报告 ——HTTP 的设计与实现



学 号 3019201274、
3019234242
姓 名 李嘉伟、曹颂
学 院 求是学部
专 业 智能机器平台
年 级 2019
任课教师 赵增华

2021 年 10 月 13 日

一、 报告摘要

基于 RFC2616 文件 http/1.1 设计文档协议，能够实现并发处理多个客户端的请求，对并发中的每一用户，能够处理其同时发出的 http pipeline 请求，并按照顺序处理，对每次请求能够做到正确封装并解析各种请求与响应、对格式错误的请求和响应进行识别，返回相应的状态码、对服务器的运行状况记录日志。

二、 任务需求分析

需要服务器能够正确解析 HEAD、GET 和 POST 方法并且做出请求资源、CGI 等正确的响应；

若同一用户并发请求，按照先后顺序依次处理并返回响应；

若请求过长，能够妥善管理服务器的接收缓冲区；

若服务器运行出现错误，能够正确处理并记录日志；

若同一用户有多条请求，能够解决粘包、同一条请求通过多次 recv 到达 server 等问题；

若有多个客户端的并发处理，能够支持 1000 个用户并发请求。

三、 协议设计

根据任务要求设计协议。包括总体设计、数据结构设计和协议规则设计。

总体设计：

在实现了 C/S 通信的基础上，须以 http 协议为基础，正确封装并解析请求与响应。具体设计包括：

客户端中须有将 echo 消息内容封装成 request 结构体的功能，以及将接收到的 response 结构体解析并提取状态码以及收到的响应正文的功能。

服务器端须有将从端口收到的 request 请求解析，识别请求方法与请求正文，并将请求正文打包成 response 结构体并发送回 client 端的功能。

所以针对 GET 和 HEAD 方法，需要的模块有：

解析器；

找到所请求的文件；

解析请求头内容（用于通知服务器请求数据的信息）；

返回响应；

记录日志；

处理同一用户的并发请求；

并发处理多个客户端的请求。

数据结构设计：

http 请求由三部分组成，分别是：请求行、消息报头、请求正文

1.请求行以一个方法符号开头，以空格分开，后面跟着请求的 URI 和协议

的版本，格式如下：

Method Request-URI HTTP-Version CRLF 其中：

Method 表示请求方法；

Request-URI 是一个统一资源标识符；

HTTP-Version 表示请求的 HTTP 协议版本；

CRLF 表示回车和换行。

2.请求方法有多种，标志着如何从服务器端操作数据。

3.请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。

4.请求头和请求正文之间是一个空行，这个行非常重要，它表示请求头已经结束，接下来的是请求正文。请求正文中可以包含客户提交的查询字符串信息。

http 请求结构范式如图 3.1 所示。

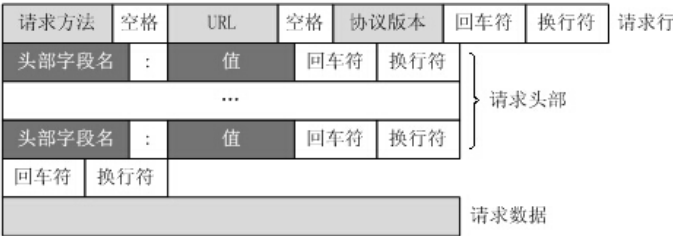


图 3.1 http 请求格式

http 响应也是由三个部分组成，分别是：状态行、消息报头、响应正文

1.状态行格式如下：

HTTP-Version Status-Code Reason-Phrase CRLF 其中：

HTTP-Version 表示服务器 HTTP 协议的版本；

Status-Code 表示服务器发回的响应状态代码；

Reason-Phrase 表示状态代码的文本描述。

状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

1xx: 指示信息--表示请求已接收，继续处理

2xx: 成功--表示请求已被成功接收、理解、接受

3xx: 重定向--要完成请求必须进行更进一步的操作

4xx: 客户端错误--请求有语法错误或请求无法实现

5xx: 服务器端错误--服务器未能实现合法的请求

2.响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对 Request-URI 所标识的资源进行下一步访问的信息。

http 响应结构范式如图 3.2 所示。



图 3.2 http 响应格式

协议规则设计：详细描述协议完成各个功能的协议规则。

http 请求方法规则：

GET 请求获取 Request-URI 所标识的资源

POST 在 Request-URI 所标识的资源后附加新的数据

HEAD 请求获取由 Request-URI 所标识的资源的消息报头

http 请求报头规则：

Accept 请求报头域用于指定客户端接受哪些类型的信息。

Authorization 请求报头域主要用于证明客户端有权查看资源。

Host 请求报头域主要用于指定被请求资源的 Internet 主机和端口号，它通常从 HTTP URL 中提取出来的。

http 响应状态码规则：

有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态代码、状态描述、说明：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的 URL

503 Server Unavailable // 服务器当前不能处理客户端的请求，一段时间后，//可能恢复正常

响应报头规则：

Location 响应报头域用于重定向接受者到一个新的位置。Location 响应报头域常用在更换域名的时候。

Server 响应报头域包含了服务器用来处理请求的软件信息。与 User-Agent 请求报头域是相对应的。

Log 设计规则:

依据 Apache 日志文件的格式, 以及实验的任务需求, 规定 Error Log Format 与 Common Log Format 的格式分别为:

Error Log

[日期和时间] [生成错误的模块/错误的严重程度(error or warning)] [进程 ID(:线程 ID)] 详细的错误信息

eg: [Fri Sep 09 10:42:29.902022 2011] [core:error]

[pid 35708:tid 4328636416] [client 72.15.99.187] File does not exist:

/usr/local/apache2/htdocs/favicon.ico

Common Log Format

主机 IP - - 日期和时间 "请求行" 状态代码 响应主体的大小(没有则置 "-")

eg: 127.0.0.1 - - [10/Oct/2000:13:55:36 -0700] "GET

/apache_pb.gif HTTP/1.0" 200 2326

四、 协议实现

1.通过递归实现多行请求

更改 parser.y 文件实现。文件中对于单个 header 的实现已经非常完备, 只需要定义 headers 结构即可。定义 headers 可以由 headers 加新 header 组成, 也可以为空。这样可以使 headers 由任意个 (0 个或有限个) header 组成。

2.Echo_server 的实现:

通过 parse 函数, 将 client 发送的缓存区中的字符串转换成 Request 结构体, 之后通过分析结构体的成员变量 http_method, 实现对 request 的识别和分析。同时需要在 echo_server 内加入对库文件 parse.h 的引用, makefile 文件中规定的编译规则也需要做相应修改。

3.socket 连接的实现

在 server 中, 调用 socket 函数建立套接字, 调用 bind 函数, 使套接字与 IP 地址和端口绑定; 调用 listen 函数, 使套接字进入被动接收状态; 使用阻塞式 accept 函数, server 将保持该状态, 直到接收到客户端的信息。在客户端, 调用 socket 函数建立套接字, 调用 connect 函数与 IP 地址和端口绑定; 绑定后, 使用 send 函数发送字符串流格式的数据包即可。

3.对 GET、HEAD、POST 方法的识别

通过 parse 函数, 将得到的字符串流请求分析后, 可以分别提取请求的各部分。分析请求的 http_method 部分, 进行字符串比较, 即可识别出 GET、HEAD、POST 方法。将请求按标准格式重新组合, 以字符串形式输出即可。

4.日志模块的实现

在每次调用响应 `response` 函数之前，首先根据响应的内容调用相应的 `Log` 函数，按照时间顺序记录下请求-响应的完成以及出现的错误。

5.实现响应同一用户的并发请求

使用 `setsockopt` 函数，更改 `socket` 输入缓存区大小，以使传入 `server` 的请求不会溢出缓存区。

在每次进行 `parse` 匹配前，清空 `parse` 缓存区，避免 `bad request` 对分析后续请求造成影响。

由于 `http1.1` 默认为 `pipelining`，故不需要对源码做太多更改，只需将接受完信息后立即断开连接改为，`connection` 头的值为 `close` 时断开连接即可。

为了避免“粘包”问题，将每次 `send` 和 `recv` 的数据大小强制定义为 `BUF_SIZE`，即 8192 字节。为每个请求增添 `content-length` 头同样可以实现这一目的，但有的请求可能并不规范，没有 `content-length` 头，故采用强制定义数据报大小的方法解决。

6.对“并发请求”功能的完善

更改缓存区中的处理方法，建立了本地的读取缓存区。每次读取后，将 `socket` 输入缓存区中的信息储存到本地缓存区中进行处理。通过寻找请求末尾的两次 `CRLF` 分隔两条请求。如果本地缓存区已满，则清空缓存区，并将未发出的请求放入新的缓存区，等待新的输入到来后合并处理。解决了之前出现的粘包、同一条请求通过多次 `recv` 到达 `server` 这些问题。

实现流程如图 4.1 所示。

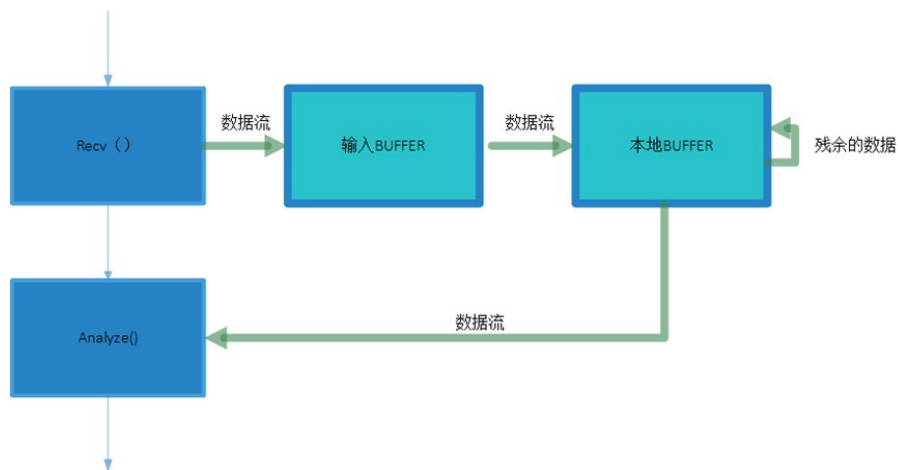


图 4.1 同一用户并发请求的实现流程

7.通过 `select()` 函数实现并发

`Select()` 实现并发的流程如图 4.2 所示。

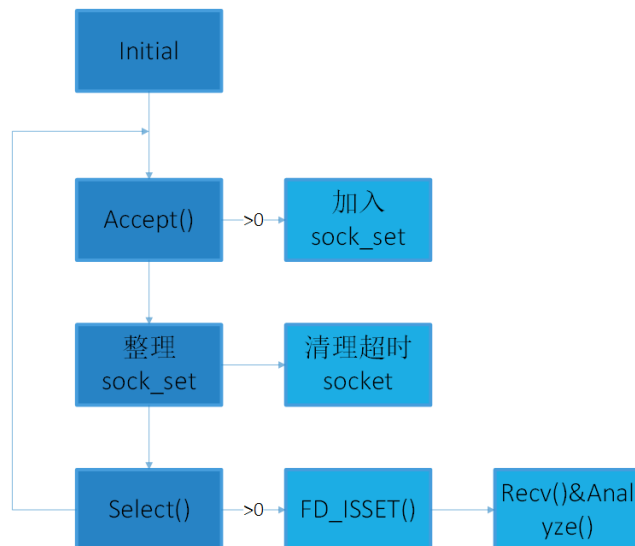


图 4.2 并发的实现流程

在监听套接字成功时,将 `accept()`接收到的连接套接字暂存入 `sock_set` 句柄集合中。

处理连接套接字时,通过 `FD_ISSET()`判断 `sock_set` 是否为空,非空时并发处理集合中的文件句柄。

一个请求结束时,分析 `Connection` 头部,在为 `close` 时判断连接结束,关闭 `socket` 连接,并将该句柄移出 `sock_set`。

需要合理地隔离处理缓存区,防止一个错误的连接污染缓存区,对之后的链接产生影响。

五、 实验结果及分析

第一周测试实验结果:

功能 1 识别 GET、POST、HEAD 方法。



图 5.1a txt 文档的内容

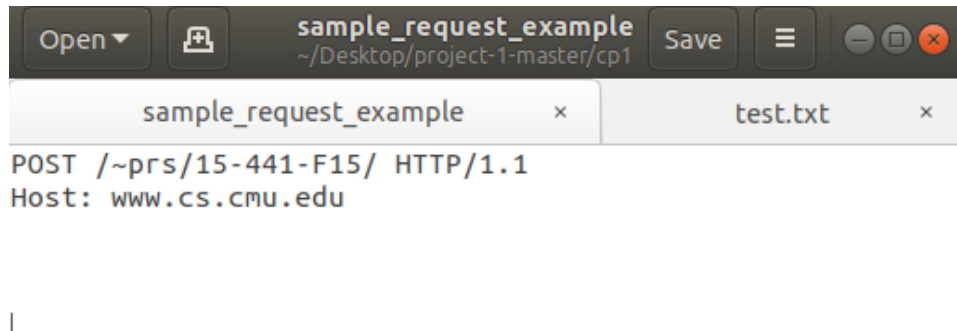
```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

Received success!!
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.1b 实验运行结果



The screenshot shows a text editor window with the title bar 'sample_request_example' and the path '~/Desktop/project-1-master/cp1'. The window contains the following text:

```

POST /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.2a 文档的内容

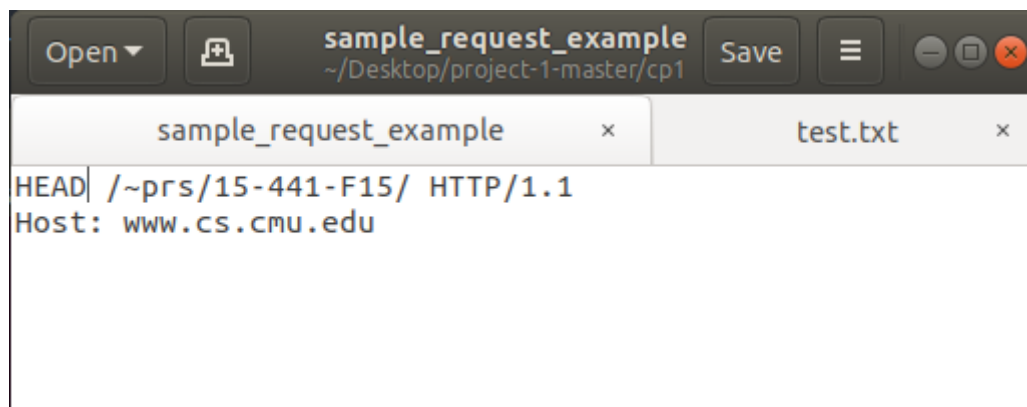
```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending POST /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

Received success!!
POST /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.2b 运行结果



The screenshot shows a text editor window with the title bar 'sample_request_example' and the path '~/Desktop/project-1-master/cp1'. The window contains the following text:

```

HEAD /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.3a 文档内容


```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_request_example
Sending HEAD /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

Received success!!
HEAD /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.3b 运行结果

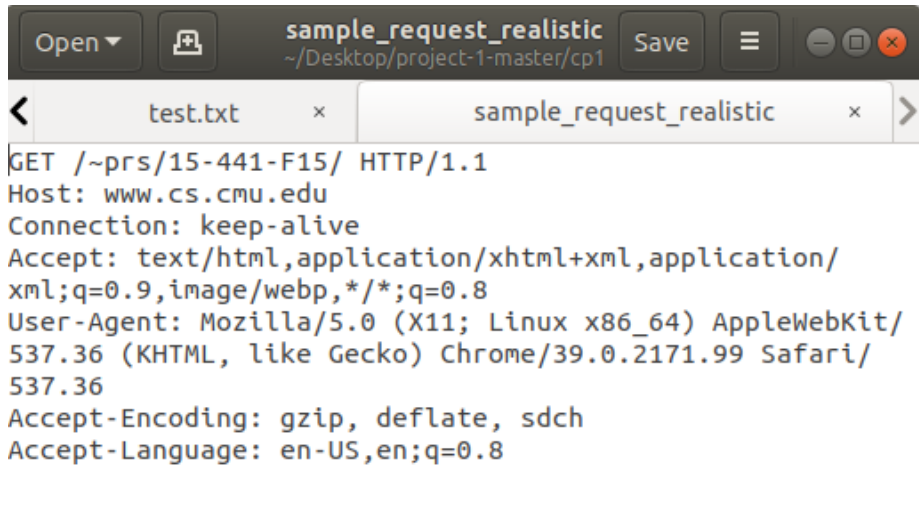


图 5.4a 文档内容

```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_request_realistic
Sending GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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

Received success!!
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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

```

图 5.4b 运行结果

功能 2 识别错误的请求并响应

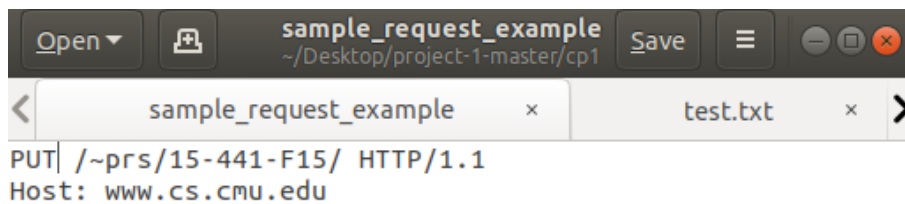


图 5.5a 文档内容

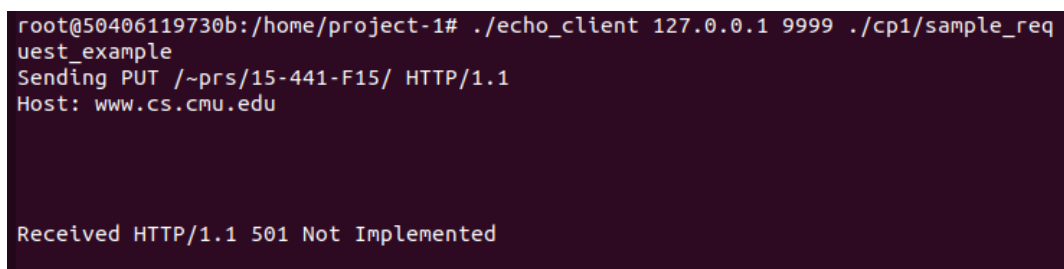


图 5.5b 测试运行结果

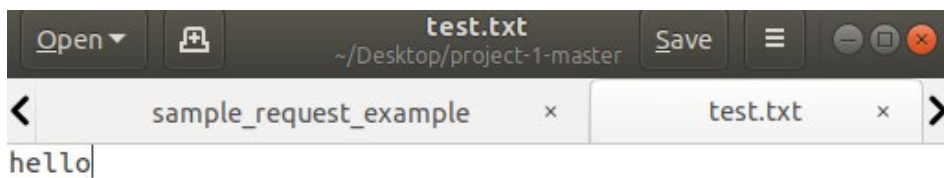


图 5.6a 文档内容

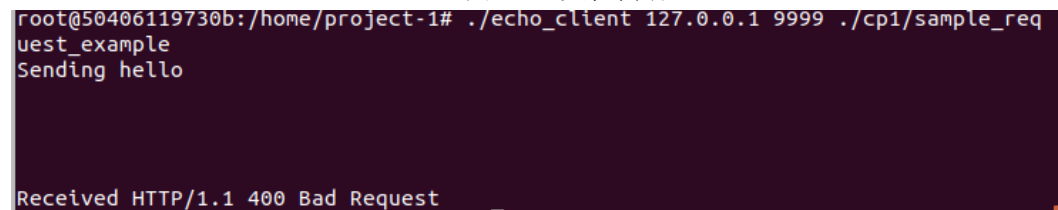


图 5.6b 实验运行结果

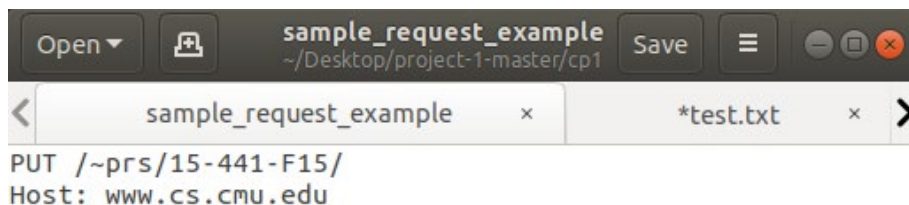
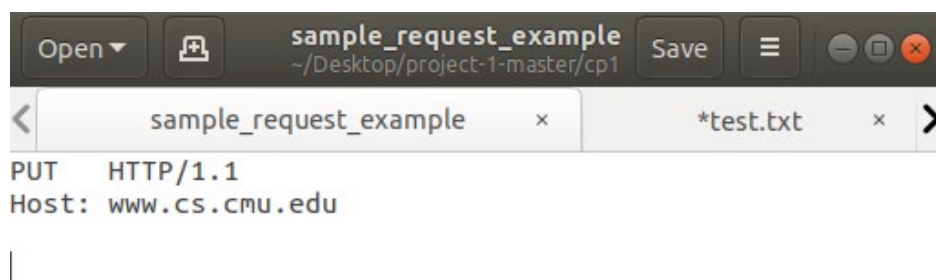


图 5.7a 文档内容

```
root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending PUT ~/prs/15-441-F15/
Host: www.cs.cmu.edu

Received HTTP/1.1 400 Bad Request
```

图 5.7b 运行结果



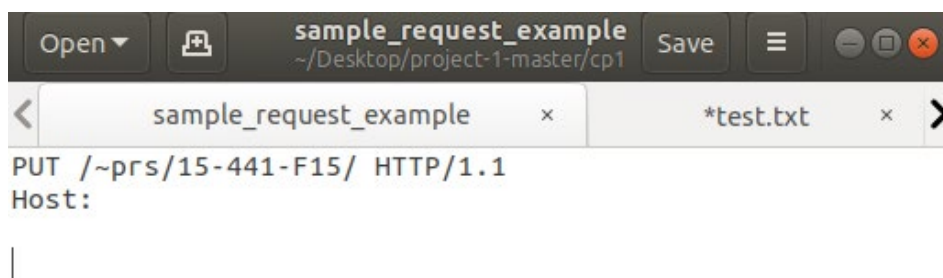
```
sample_request_example
~/Desktop/project-1-master/cp1
Save
sample_request_example x *test.txt x
PUT HTTP/1.1
Host: www.cs.cmu.edu
|
```

图 5.8a 文档内容

```
root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending PUT HTTP/1.1
Host: www.cs.cmu.edu

Received HTTP/1.1 400 Bad Request
```

图 5.8b 运行结果



```
sample_request_example
~/Desktop/project-1-master/cp1
Save
sample_request_example x *test.txt x
PUT ~/prs/15-441-F15/ HTTP/1.1
Host:
|
```

图 5.9a 文档内容

```
root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending PUT ~/prs/15-441-F15/ HTTP/1.1
Host:

Received HTTP/1.1 400 Bad Request
```

图 5.9b 运行结果

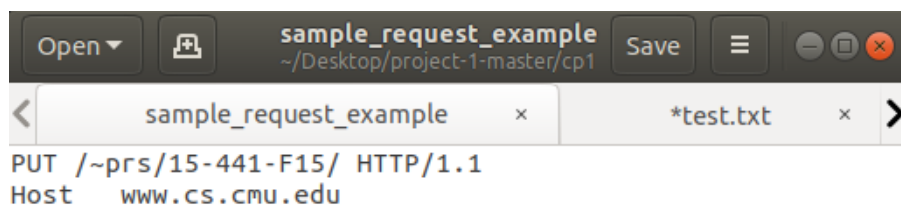


图 5.10a 文档内容

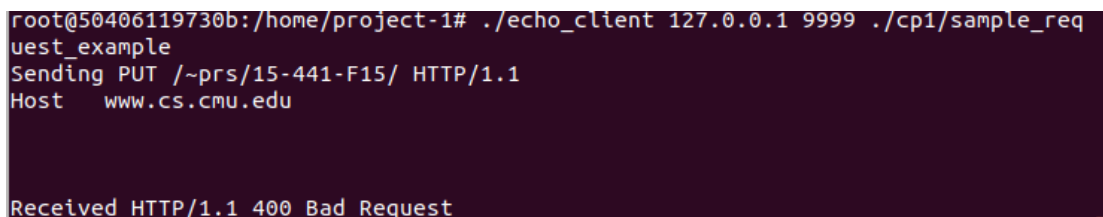


图 5.10b 运行结果

第二周实验测试样例：

各种出错状态码：

400 Bad Request 运行结果如图 5.11 所示。

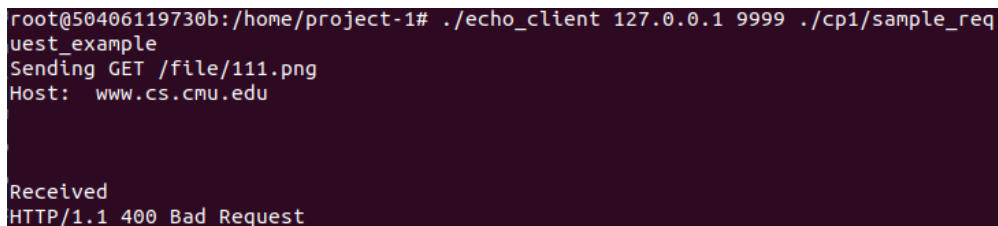


图 5.11 400 状态码

404 Not Found 运行结果如图 5.12 所示。

使用 access 函数实现，如果文件不存在或不可读，则错误。

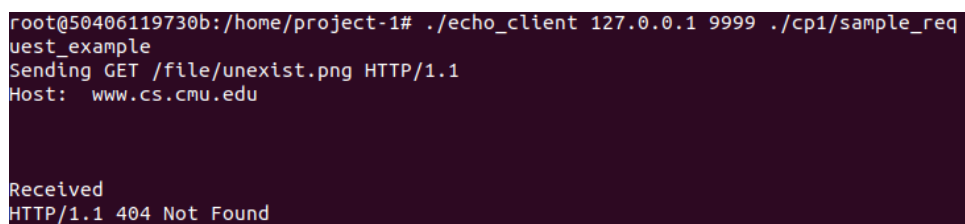


图 5.12 404 状态码

408 Request Timeout 测试中将 socket 改为非阻塞模式，这样就可以在循环调用 accept 的同时计时，计算从建立链接到接收到信息的时长，本次实验中将等待时间限制在 1s。在 client 端通过 sleep 函数，让 client 在连接套接字之后，睡眠 5 秒钟再发送 request。运行结果如图 5.13 所示。

```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
sleep for 5 seconds
Sending
GET /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Received
HTTP/1.1 408 Request Timeout

```

图 5.13 408 状态码

501 Not Implemented 运行结果如图 5.14 所示。

```

root@50406119730b:/home/project-1# ect-1# client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending
PUT /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Received
HTTP/1.1 501 Not Implemented

```

图 5.14 501 状态码

505 Version Not Supported 运行结果如图 5.15 所示。

```

root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_req
uest_example
Sending
GET /file/111.png HTTP/1.0
Host: www.cs.cmu.edu

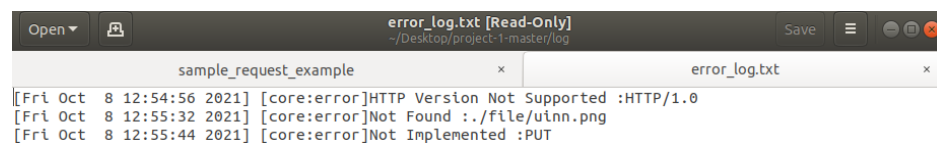
Received
HTTP/1.1 505 HTTP Version Not Supported

```

图 5.15 505 状态码

日志记录功能

Error Log 输出样例如图 5.16 所示。



```

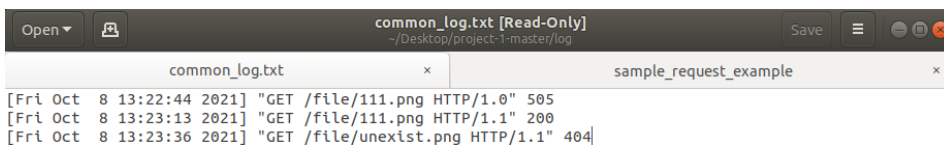
error_log.txt [Read-Only]
~/Desktop/project-1-master/log

sample_request_example x error_log.txt x
[Fri Oct 8 12:54:56 2021] [core:error]HTTP Version Not Supported :HTTP/1.0
[Fri Oct 8 12:55:32 2021] [core:error]Not Found :./file/uinn.png
[Fri Oct 8 12:55:44 2021] [core:error]Not Implemented :PUT

```

图 5.16 Error Log

Common Log 输出样例如图 5.17 所示。



```

common_log.txt [Read-Only]
~/Desktop/project-1-master/log

common_log.txt x sample_request_example x
[Fri Oct 8 13:22:44 2021] "GET /file/111.png HTTP/1.0" 505
[Fri Oct 8 13:23:13 2021] "GET /file/111.png HTTP/1.1" 200
[Fri Oct 8 13:23:36 2021] "GET /file/unexist.png HTTP/1.1" 404

```

图 5.17 Common Log

第三周实验测试样例：

连续向服务器发送 20 次并发的请求，结果如图 5.18 所示。

```
root@50406119730b:/home/project-1# ./echo_client 127.0.0.1 9999 ./cp1/sample_request_example
Sending
GET /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:2
GET /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:3
GET /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:4
POST /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:5
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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
```

图 5.18a 并发请求

```
root@50406119730b:/home/project-1
File Edit View Search Terminal Help

Sending:6
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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

Sending:7
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive

Sending:8
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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

Sending:9
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
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
```

图 5.18b 并发请求

```

Sending:10
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:11
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:12
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:13
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:14
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:15
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

Sending:16
HEAD /file/111.png HTTP/1.1
Host: www.cs.cmu.edu

```

图 5.18c 并发请求

第四周实验测试样例：

使用 `siege` 进行并发测试，并发发送 1000 条 GET 请求，图 5.19 为测试输出结果，并发数 1000，成功率 100%。

```

sudo ./echo_server
File Edit View Search Terminal Help
t:token_char e
t:token_char c
t:token_char t
t:token_char i
t:token_char o
t:token_char n
t:colon;
t:sp ' ';
t:token_char c
t:token_char l
t:token_char o
t:token_char s
t:token_char e
t:crlf;
t:crlf;
analyze-parse succ
analyze-GET compare succ
strcat suc:./file/111.png
send respond 1
analyze-GET compare end
analyze-send succ
cclose
sock 0 closed by client
_client
_server
ple

sant1an@ubuntu:~/Desktop/project-1-master
File Edit View Search Terminal Help
[oh-my-zsh] plugin 'git,' not found
~/Desktop/project-1-master > siege -c 1000 -r 1 127.0.0.1:9999/file/111.png
** SIEGE 4.0.4
** Preparing 1000 concurrent users for battle.
The server is now under siege...
Transactions:      1000 hits
Availability:      100.00 %
Elapsed time:      7.39 secs
Data transferred:  0.00 MB
Response time:     1.65 secs
Transaction rate:  135.32 trans/sec
Throughput:        0.00 MB/sec
Concurrency:       222.66
Successful transactions: 1000
Failed transactions:  0
Longest transaction: 7.38
Shortest transaction: 0.01
~/Desktop/project-1-master > aom ~/.siegerc
7s 03:37:21

```

图 5.19 siege 输出结果