**《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**

1. **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。

1. **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。

1. **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。

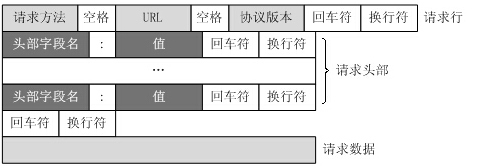


图3.1 http请求格式

**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。

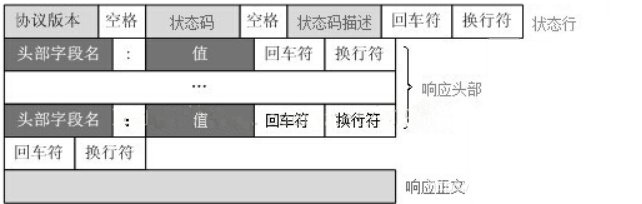


图3.2 http响应格式

**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

1. **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。这样可以让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。

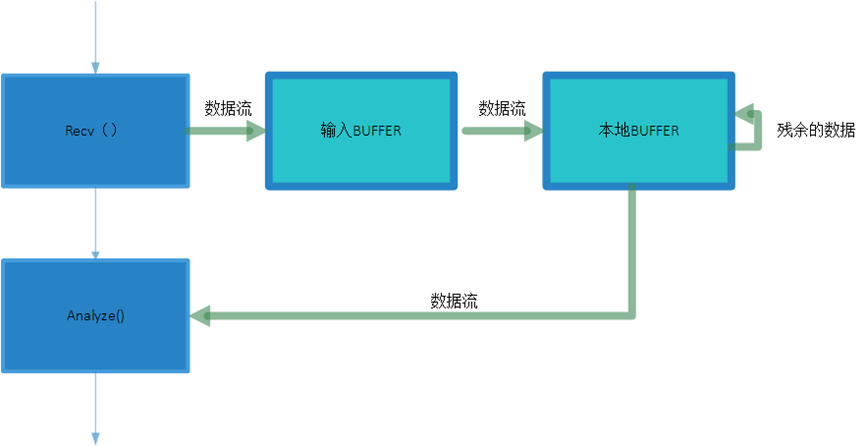


图4.1 同一用户并发请求的实现流程

7.通过select()函数实现并发

Select()实现并发的流程如图4.2所示。

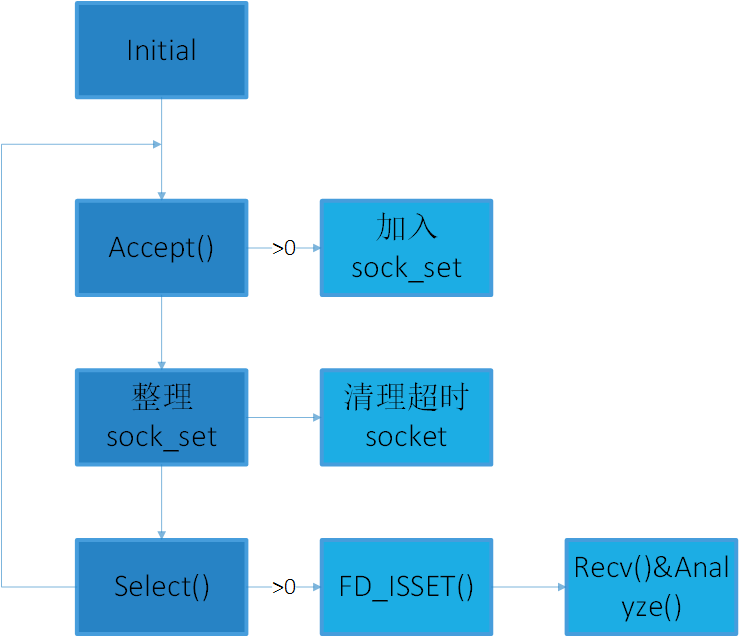


图4.2 并发的实现流程

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。

1. **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

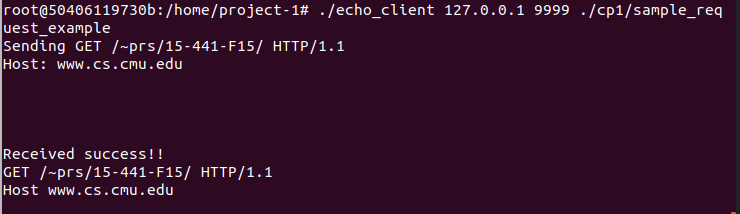


图5.1b 实验运行结果

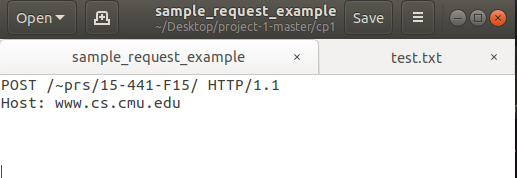


图5.2a 文档的内容

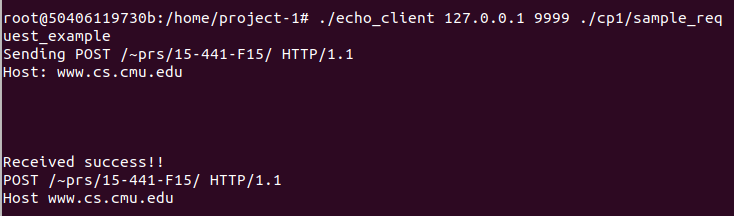


图5.2b 运行结果

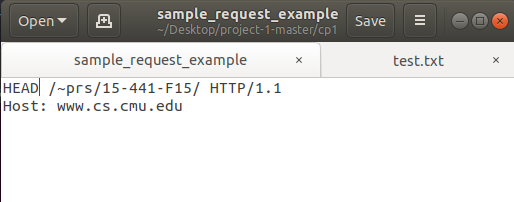


图5.3a 文档内容

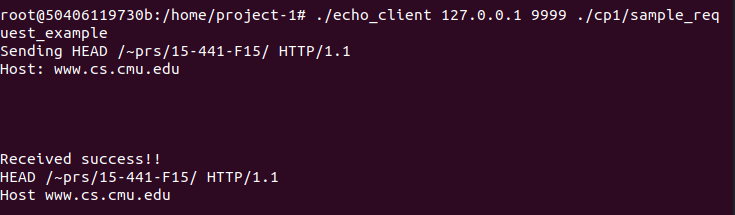


图5.3b 运行结果

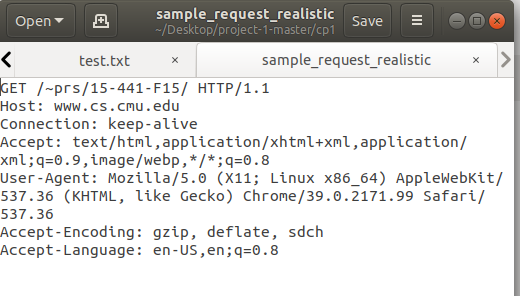


图5.4a 文档内容



图5.4b 运行结果

功能2 识别错误的请求并响应

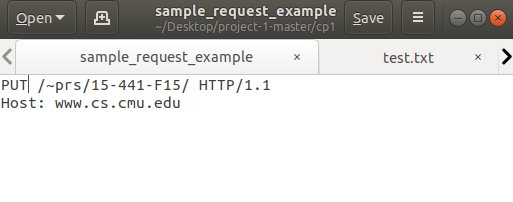


图5.5a 文档内容

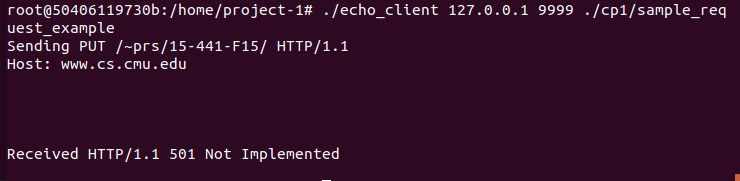


图5.5b 测试运行结果

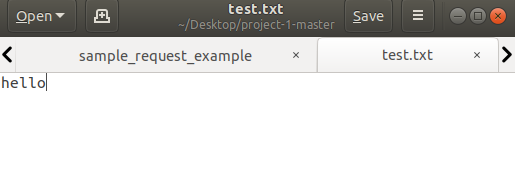


图5.6a 文档内容

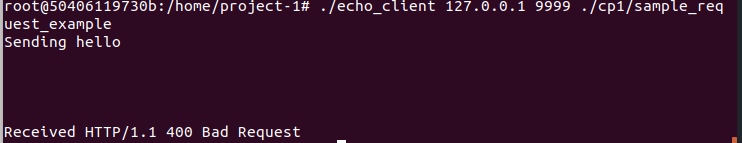


图5.6b 实验运行结果

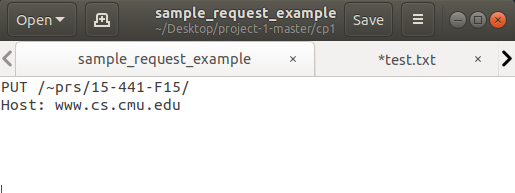


图5.7a 文档内容

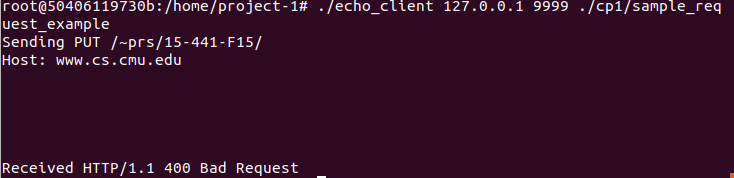


图5.7b 运行结果

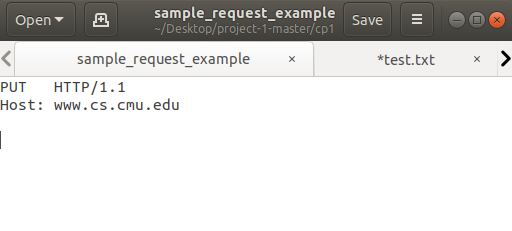


图5.8a 文档内容

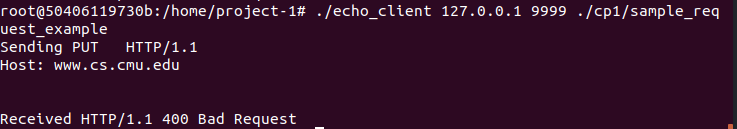


图5.8b 运行结果

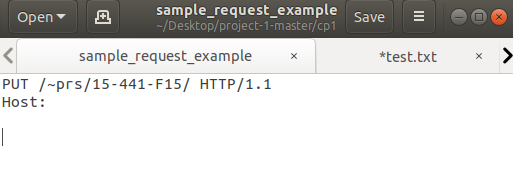


图5.9a 文档内容

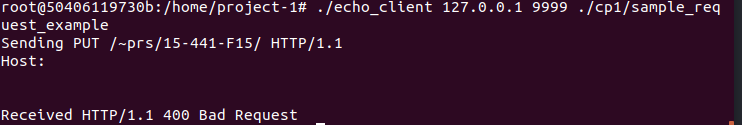


图5.9b 运行结果

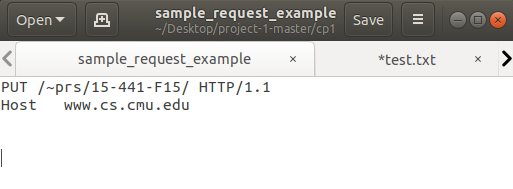


图5.10a 文档内容

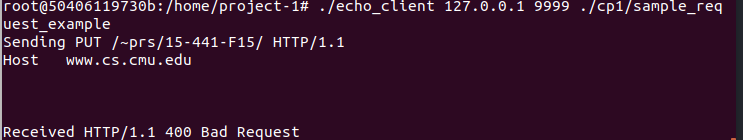


图5.10b 运行结果

**第二周实验测试样例：**

各种出错状代码：

400 Bad Request运行结果如图5.11所示。

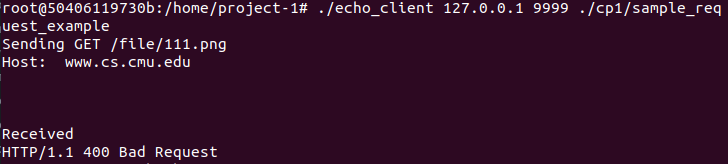


图5.11 400状态码

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。

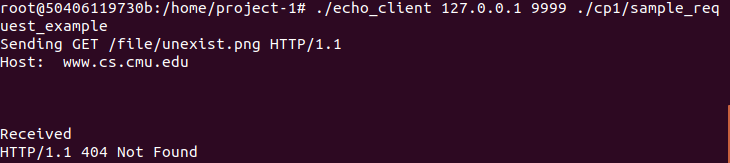


图5.12 404状态码

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。

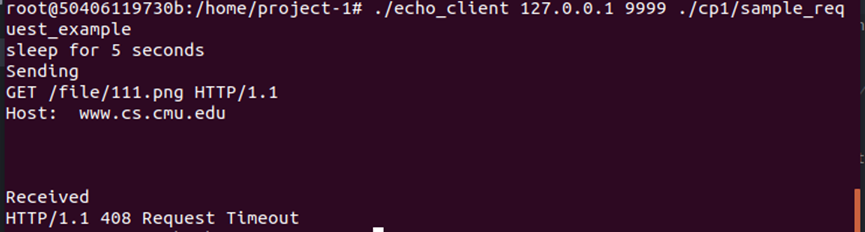


图5.13 408状态码

501 Not Implemented运行结果如图5.14所示。

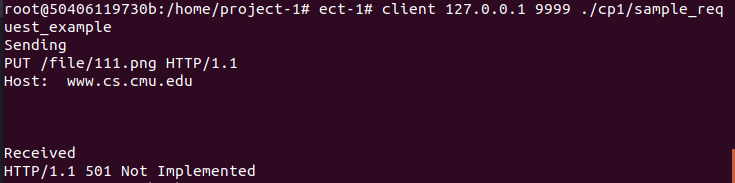


图5.14 501状态码

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

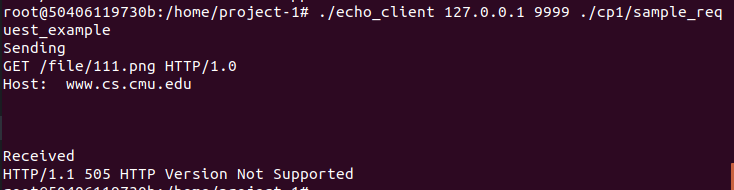


图5.15 505状态码

**日志记录功能**

Error Log输出样例如图5.16所示。

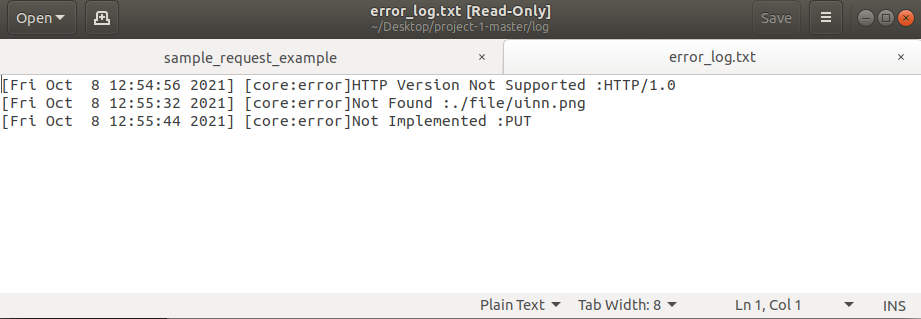
****

图5.16 Error Log

Common Log输出样例如图5.17所示。

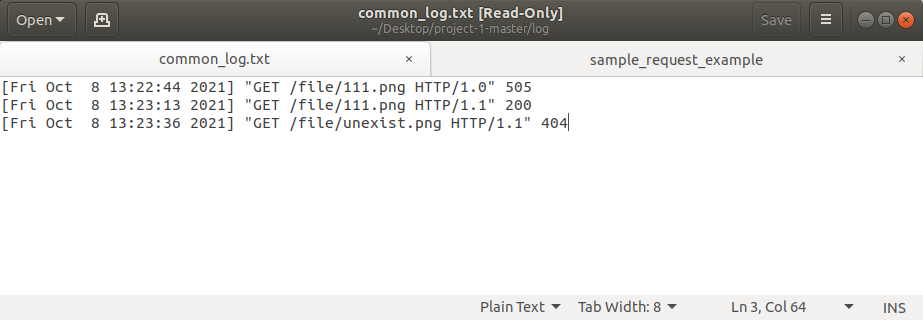


图5.17 Common Log

**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。

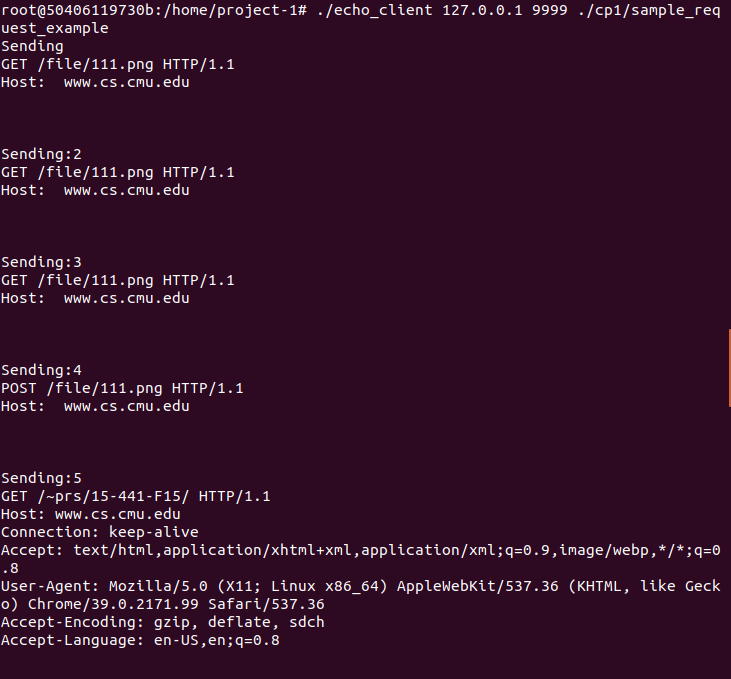


图5.18a 并发请求

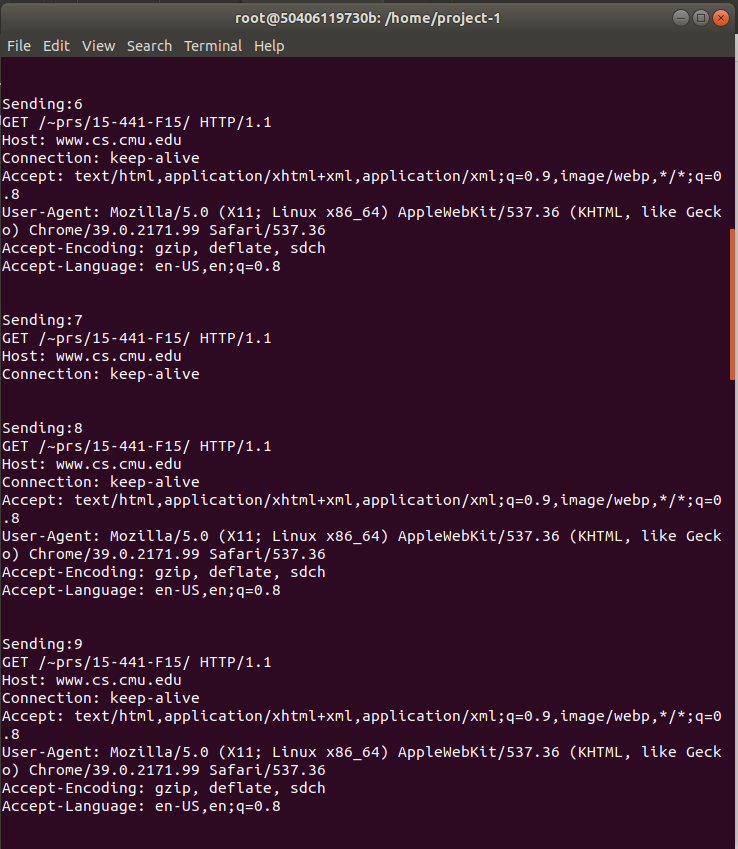


图5.18b 并发请求

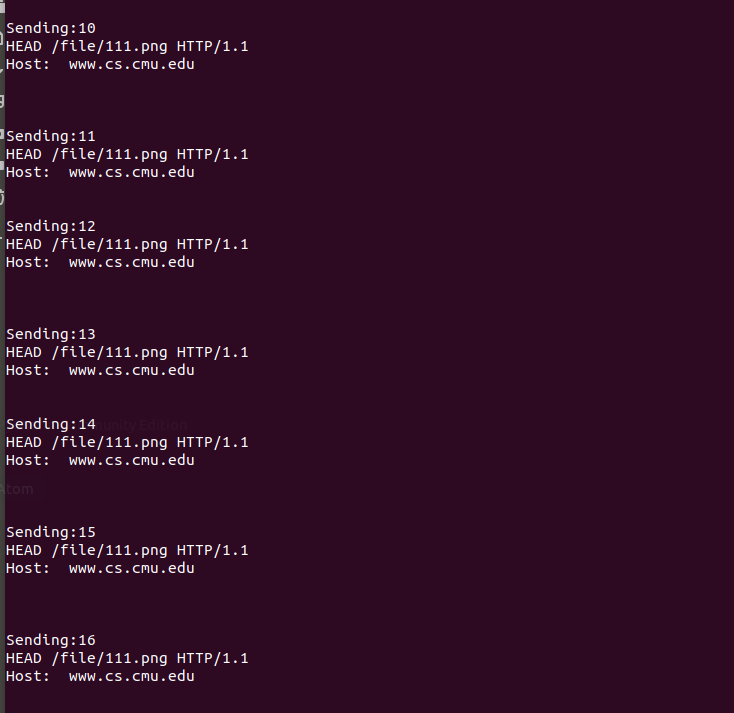


图5.18c 并发请求

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%。

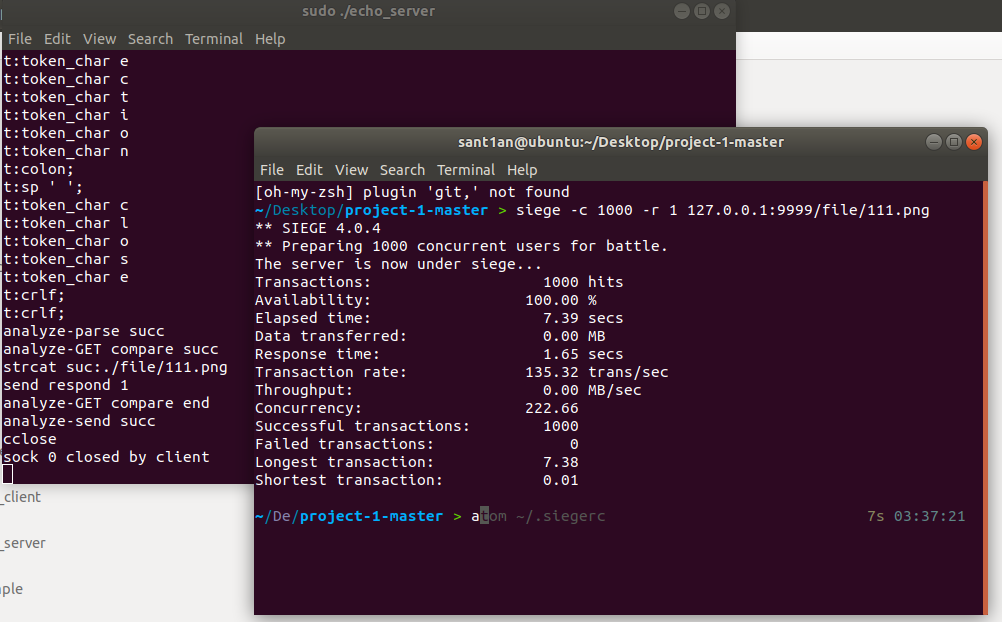


图5.19 siege输出结果