**《计算机网络》课程设计报告**





**Web Server的设计与实现**

**学 号 3523058921 3523058920**

**姓 名 高全乐易 向菲**

**学 院 智能与计算学部**

**专 业 计算机科学与技术**

**年 级 2022级**

**任课教师 赵增华**

**2024年5月19日**

# 一、报告摘要

（简要介绍需要解决的具体问题、协议设计和实现，以及主要实验结果。）

本实验主要需要搭建docker下的编程环境，根据HTTP协议，Socker编程方法，Lex和yacc正确解析消息的方法实现简单的echo web server，在Server收到client的多条消息时能正确解析并且返回对应的响应信息。

# 二、任务需求分析（在第一周完成这部分内容）

（分析实践任务需要解决的具体问题。）

**任务需求**：

该实验需要使用BSD Socket ABI实现具有并发性的web server。主要分为四个部分，实现GET，HEAD，POST三种基本方法，正确解析客户端发送的请求包同时做出正确的响应，支持HTTP的并行请求，支持多个客户端并发接入。

**具体问题：**

1. 熟悉Socker编程，lex和yacc正确解析消息的问题，实现简单的echo web server。
2. 完善服务器功能，正确响应HTTP1.1请求消息，实现HEAD，GET，POST的持久连接；创建简化的日志记录模块，记录格式化日志。
3. 对于服务器利用RFC2616规定的顺序处理HTTP的并发请求，支持HTTPpipelining。
4. 实现服务器在等待下一个请求时，能同时处理其它客户端的请求，实现并行化。

# 三、协议设计

**3.1 总体设计（**在第一周完成这部分内容）

（根据任务分析划分功能模块，例如请求响应模块、日志模块、并发处理模块等，理清模块组织结构和关系等。）

该实验主要可以划分为以下模块：

1. **请求响应模块：**主要负责将客户端发送的HTTP请求解析为服务器内部可以处理的数据结构。该实验中主要体现为parser模块，包括example.c，parse,c，parse.y等文件，他们对客户端的HTTP请求进行解析，将处理结果封装成一个响应对象发送回客户端，响应结果包括网页内容，状态码，响应头等。
2. **日志模块：**可以记录服务器的运行信息，包括对应的请求，响应，错误等运行日志，访问日志，错误日志，便于调式的日志等等，可以用于故障排除和性能优化。
3. **并发处理模块：**用select()的方式实现HTTP的并发请求。也就是服务器支持HTTPpipelining，并且等待下一个客户端发送下一个请求时，能同时处理其它客户端的请求，实现并发处理提高性能。
4. **其它模块：**包含客户端服务端程序，业务逻辑处理等基本内容。其中包含以下：

对客户端请求的相应处理，包括静态文件的读取等

Echo\_server:接收客户端发送的数据并将其原样返回客户端，通常用于测试和调试，检测网络是否正常工作。

Echo\_Client:客户端向服务器发送数据，等待服务起返回相同的数据，通常时用户或其他应用程序。

Webserver中请求响应模块是核心模块，日志模块用于记录日志排查故障，并发处理模块实现对于多客户端的处理提升性能等等，不同模块各司其职，协同工作，确保服务器能够高效的处理客户端并提供相应。

## **3.2 简单echo web server的设计**

**（**从本节开始，下面每节对应一周的设计内容。每节都从数据结构设计、协议规则设计两个方面描述所做的设计，数据结构设计需要详细描述所设计协议的数据结构。包括协议头部结构、主要的数据结构等。协议规则设计需要详细描述协议完成各个功能的协议规则。）

1. **数据结构设计：**

协议头部结构：遵循HTTP/1.1标准，包含请求行、请求头部和消息体等部分。头部字段包含：Method，URL，Version，StatusCode。

主要数据结构：Request结构体，其中包含http\_version,http\_method,http\_url;

1. **协议规则设计:**

接收消息规则：

服务器(server)使用listen()进行监听等待客户端(client)的连接请求，然后使用accept()获取客户端的连接请求并建立连接，建立好连接后，通过套接字描述符进行收发数据。接收到客户端HTTP请求信息后，按照HTTP协议规则解析请求信息，并存入Request结构体中。

发送消息规则：

客户端(client)使用socket()函数建立套接字，使用connect()函数请求与服务器建立连接。建立好连接后，可通过套接字描述符，send(),recv()函数进行收发数据。

服务器处理信息规则：

对于从客户端(client)接收到的HTTP请求进行解析，如果收到GET，POST，HEAD三个请求中的任意一个，服务器(server)重新封装消息并返回给客户端(client)。

如果接收到非GET，POST，HEAD方法，服务器(server)将返回响应信息 "HTTP/1.1 501 Not Implemented\r\n\r\n"。

如果收到的请求信息错误，如无法解析，请求头部存在格式问题，服务器(server)将返回响应信息"HTTP/1.1 400 Bad request\r\n\r\n"。

通过上述的数据结构以及规则的设计，一个简单的echo web server能够完成接受客户端多请求行消息，并进行正确的解析，返回相应的响应信息的功能。

## **3.3 基本HEAD、GET、POST方法的设计**

1. HTTP1.1三种基本方法的设计规则和协议头头部格式

**HEAD**

HEAD的请求常用于请求一个GET请求相同的响应，但是不包括响应体也就是返回请求的资源内容，该请求常用于返回资源的相关信息，比如资源类型，内容长度，状态码等。这样有利于判断其大小决定是否值得下载，状态码用于测试资源的可用性等。

**GET**

GET的请求用于请求制定的资源。使用GET方法时，请求参数通常附加在URL之后，通过URL查询字符串传递给服务器。GET请求只应当用于获取数据，不应用于修改数据。同时GET请求中的URL具有长度限制，请求参数不能过长。

**POST**

POST请求用于向服务器提交数据，通常用于创建或者更新资源，它的请求的参数通常不是在URL上，而是包含在请求体上，理论上请求参数和请求体的大小没有限制，但实际上可能受到服务器处理能力和配置的限制，该请求常用于提交表单数据，上传文件等。

2．接收缓冲区

除了在第一周对客户端和服务端的接收缓冲区和发送缓冲区之外，在本周对于GET的请求中文件的长度不宜太大超过8192，因此设置了缓冲区file[BUF\_SIZE]，先将文件中的数据存储在缓冲区中，再将其发送回客户端。

1. 日志记录模块

在该模块中，参考Apache日志的文件进行设计，主要分为三部

log\_init日志初始化,log\_access和log\_error。

Log\_access访问日志

记录了所有对服务器的访问活动。该日志对于了解网站情况，访问者的行为模式，服务器性能等方面效果很好。访问日志通常包含请求的远程主机名或者IP地址，请求时间，请求方法，请求的协议版本，响应的状态码，发送给用户端的字节数等。

Log\_error错误日志

记录了服务器运行期间遇到的各种错误以及诊断信息。这些错误可能是由服务器配置，客户端请求，文件打开错误等问题，错误日志通常包含错误发生的日期和事件，错误的严重程度，描述问题性质的错误消息等等。

## **HTTP Pipelining的设计**

**1.数据结构设计：**

服务器端(server):

建立了新的字符型数组new\_request用来存放提取出来的数据。

使用了两个指针，其中一个指针指向接受消息的起始位置，令一个指针指向分隔符所在位置。中间的文件就是所提取出来的请求消息

客户端(client):

使用了两个指针，其中一个指针指向接受消息的起始位置，令一个指针指向分隔符所在位置。中间的文件就是所提取出来的请求消息

**2.协议规则设计**

在HTTP Pipelining中，客户端可以连续发送多个请求，不同的请求之间用\r\n\r\n分隔。服务端接收到多个请求后，按照它们被接收的顺序依次处理。

## **3.5 多个客户端的并发处理的设计**

**1.数据结构设计：**

Client\_sockets[MAX]数组，用于存放文件描述符。

**2．协议规则设计：**

基本逻辑就是在一个while（1）循环里面不停地轮询 accept() 和 select()函数。

使用select()函数和fd\_set文件描述符集合，可以实现多个客户端的并发处理。服务器在调用select()后等待，直到被监视的文件描述符中有一个或多个发生状态改变。通过创建空的fd\_set集合来管理需要监视的文件描述符，将服务器的监听套接字添加到集合中。

当接收到客户端连接时，将新的客户端套接字添加到fd\_set集合中。然后，服务器再次调用select()函数等待，以监视所有文件描述符的状态。通过使用FD\_ISSET()来检查哪个文件描述符准备好读取，服务器可以处理相应的客户端请求。

在并发处理中，即使一个客户端暂停或出错，也不会影响其他并发用户。服务器在处理每个客户端请求时进行适当的错误处理，确保不被阻塞。服务器处理完一个客户端请求后，可以再次使用select()函数等待下一个就绪的文件描述符，实现循环处理多个并发客户端的请求。通过适时地清除和设置fd\_set集合中的文件描述符位，服务器能够动态管理多个客户端的连接。

**3.6 CGI的设计（选做）**

# 四、协议实现

（详细描述功能实现的细节。主要功能模块使用流程图或者伪代码来辅助说明。禁止贴源码。）

## **4.1 简单echo web server的实现**

**（**从本节开始，下面每节对应一周的实现内容）

**WEEK1 任务分工：**

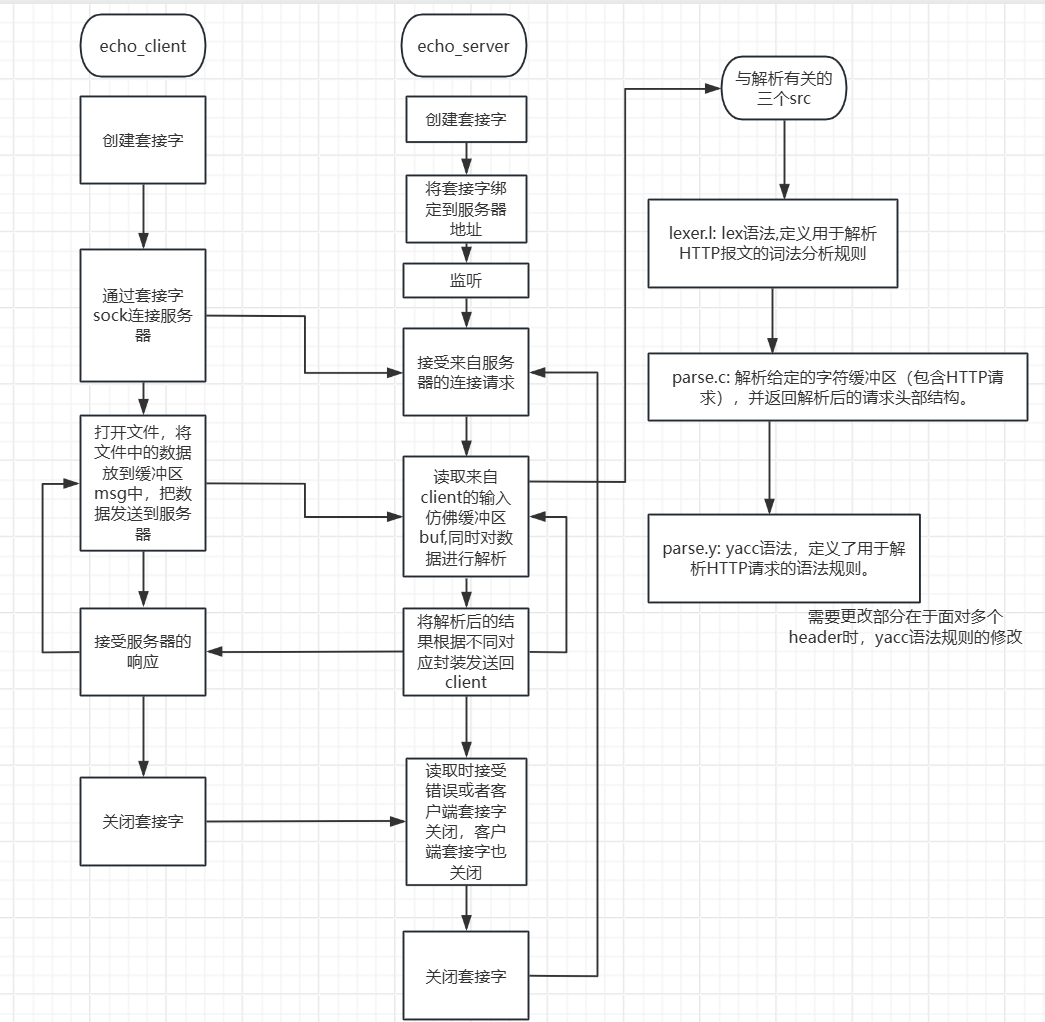
**高全乐易完成client.c文件的编写**

**向菲完成server.c文件的编写**

**共同完成Lex，yacc解析器的分析以及parse.c文件,makefile的修改。**

在该实验中，大致内容为：server启动服务器和端口号绑定开始监听，然后由client发送数据，server为其解析接受并根据请求的差异进行不同的发送，最后由client收到server发送的数据并输出。

具体流程图如下：



在整个流程中，关键在于对客户段传来的数据进行解析同时进行恰当的返回。对数据的接收采用recv函数，对数据的解析依赖于lexel.c,parse.y和parse.c三个函数，前两主要用于定义HTTP报文的词法分析和语法规则，只针对特定的语法规则进行解析并且传到机构体request中，parse.c函数就是对语句利用规则进行解析，将接收到的合适值传到request指向的http\_method中，同时对客户段的请求返回合适的数据或者501或者400。

## **4.2 基本HEAD、GET、POST方法的实现**

**WEEK2任务分工：**

**高全乐易完成：完成日志格式文件函数编写**

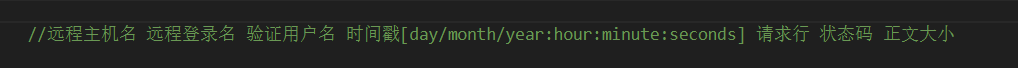
**向菲完成：完成liso\_server.c内部相关函数编写**

**4.2.1日志记录**

日志记录模块基于Apache标准进行设计，在文件中创建了log.c和log.h两个文件，log.c文件中封装了log\_acc()文件和log\_err()文件，分别用来处理访问日志记录和错误日志记录。

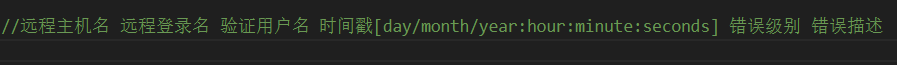
**访问日志记录协议实现：**

1. 在服务器端调用log\_acc()文件时，使用内置的time相关函数记录下当前的时间，该函数会获取当前时间、客户端IP、请求信息、响应状态码和响应大小等信息并记录下来。
2. 函数会将上述记录下的信息按照Apache的log格式标准存入到访问日志中，根据Apache标准日志记录格式应为：



**错误日志记录协议实现：**

1. 当发生错误如文件不存在，请求头错误时，服务器端会调用log\_err()函数，同样使用内置的函数获取当前时间，客户端IP，错误级别和错误描述并记录下来
2. 函数会将上述记录下的信息按照Apache的error log标准格式存入到错误日志中，根据Apache标准，错误日志记录格式应为：



**4.2.2 HEAD，GET，POST方法的实现**

每个返回的内容主要由三部分组成。第一部分是响应状态行，其中包括HTTP 版本（version）状态码和响应信息。第二部分是头部信息，包括响应内容的格式类型（content\_type）、文件长度（content\_length）、产生时间等相关信息。最后是响应体，可以返回请求的文件内容。

处理相应请求的伪代码如下：

接收请求函数get\_request:

    调用parse函数解析请求得到请求方法,url,HTTP version

    清空缓冲区

    IF 解析为空 :

        返回400响应

        记录到访问日志

    IF HTTP version 不为 HTTP/1.1 :

        返回505响应

        记录到访问日志

    IF 请求方法为 GET :

        调用处理GET方法函数

    IF 请求方法为 HEAD :

        调用处理HEAD方法函数

    IF 请求方法为 POST :

        POST重新封装返回

    ELSE :

        返回501相应

        记录到访问日志

处理GET方法函数 request\_get:

    检查文件状态

    IF 文件状态为不存在或无权限访问:

        返回400响应

        并记录到错误日志中

    ELSE :

        返回200响应

        将文件状态记录为headers格式,并返回

        返回文件内容

        记录到访问日志中

处理HEAD方法函数 request\_get:

    检查文件状态

    IF 文件状态为不存在或无权限访问:

        返回400响应

        并记录到错误日志中

    ELSE :

        返回200响应

        将文件状态记录为headers格式,并返回

        记录到访问日志中

## **4.3 HTTP Pipelining的实现**

**WEEK3任务分工：**

**高全乐易完成服务器端（server）的流水线接受处理。**

**向菲完成客户端（client）的流水线接受响应。**

为了实现pipeline处理接受请求和进行相应，我们在上次实验的基础上对get\_request()函数进行了循环操作，使其能够接收到多个请求，为了将每个请求分割开来进行相应的处理，我们使用两个指针，一个指针指向文件开头，一个指针指向分隔符，借此来将多个请求分离成单个请求，并逐一进行操作。指向文件开始的指针在进行完一次请求的处理后，会移动到分隔符所在的指针下面进行第二次的接受，同理，另一个指针会移动到下一个分隔符。以此类推，直到指向分隔符的指针为NULL时，说明流水线请求已经被全部处理完成。客户端同理，对接收到的msg进行分割处理，将不同请求的不同响应分割出来，最后输出。

伪代码如下：

设置指针P 指向文件开头

设置指针pos 指向分隔符

初始化字符数组 new\_request  用来存储提取到的数据

while pos指针不为空时

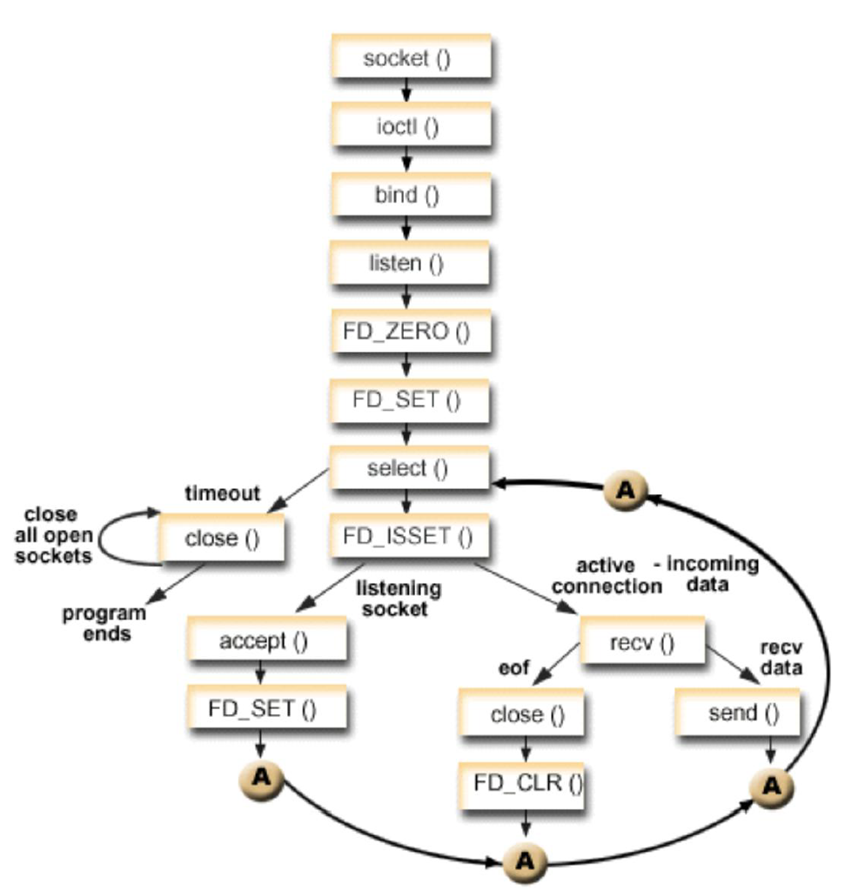
    将两个分隔符之间的数据拷贝到new\_request中作为接收到的请求

    调用get\_request()函数进行处理new\_request中的数据

    更新指针p指向下一个起始位置

    更新指针pos指向下一个分隔符

## **4.4 多个客户端的并发处理**



只需在liso\_server.c文件中修改while(1)内容即可，但是需要先创建文件描述符集合，并且将服务器监听的主套接字加入到文件描述集合中，然后使用循环遍历服务爱的客户端套接字，加入到集合中，更新最大的文件描述符值。

Select()函数的详细用法是select(max+1,&readfds,NULL,NULL.&timeout)。其中max是最大的文件描述符值，readfds是文件描述符集合，NULL表示不监视写和异常时间，timeout是超时参数。之后调用FD\_ISSET()函数判断是否有新的连接请求，并将其加入数组中，最后只需连理服务器管理的客户端套接字并进行accept和recv，然后循环调用get\_request()函数对于客户端的请求进行处理即可。

FD\_ZERO()->readfds

FD\_SET()

for i : 0 -> MAX\_CLI

    获取客户端的套接字描述符

    若有效则加入readfds

ready = select(max\_fd+1,readfds,NULL,NULL,timeout)

IF ready < 0

    select failed

IF ready == 0

    time out

for a : 0 -> MAX\_CLI

    FD\_ISSET() : sock 在 readfds 中存在

        进入内部循环

            不断调用get\_request()接受请求

**4.5 CGI的实现（选做）**

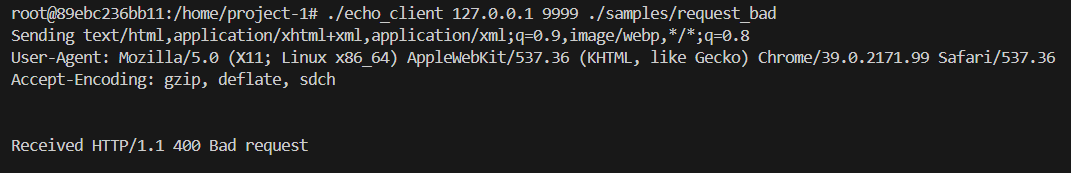
# 五、实验结果及分析

（测试所实现协议的功能和性能，并对性能结果进行分析。需要针对考察点逐一展开。下面每节对应一周的内容）

## **5.1 简单echo web server的实验结果与分析**

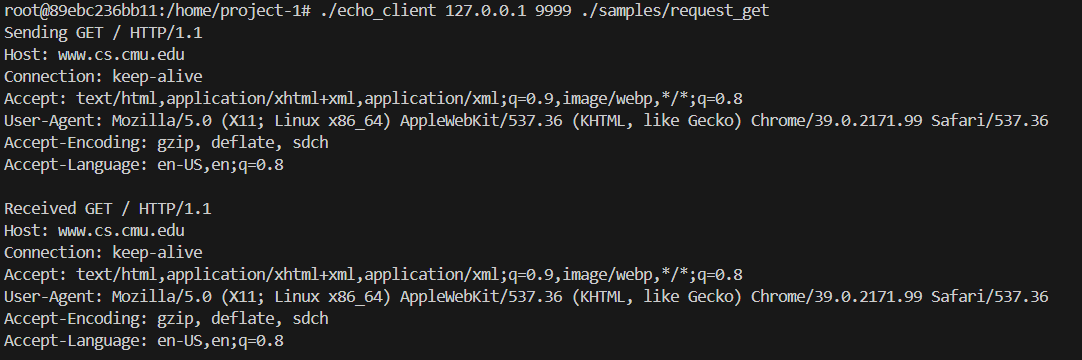
使用request\_bad文件模拟请求信息错误的情况

返回"HTTP/1.1 400 Bad request\r\n\r\n"



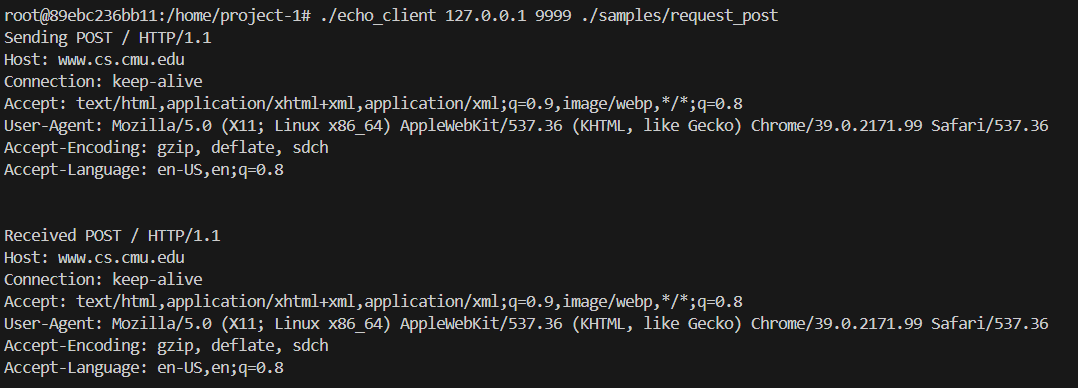
使用request\_get文件模拟GET请求头

返回重新封装的请求信息。



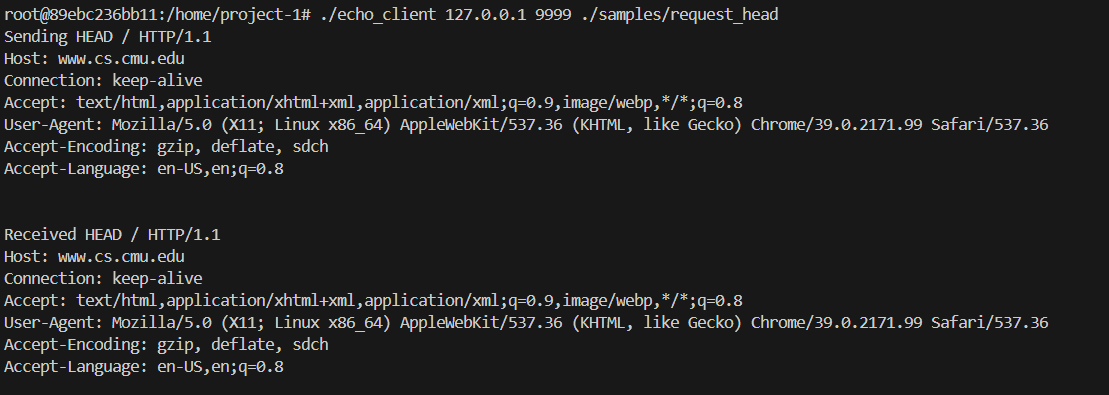
使用request\_post文件模拟POST请求头

返回重新封装的请求信息。



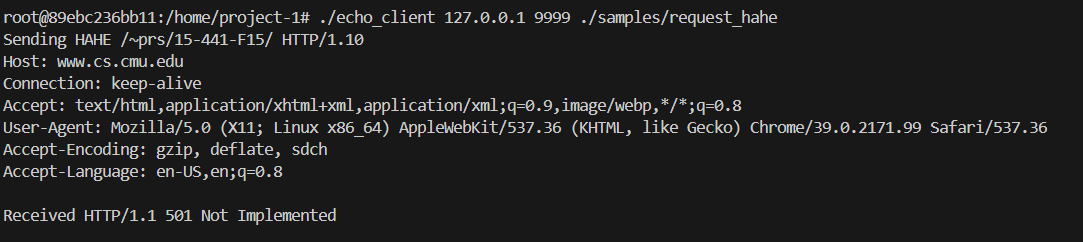
使用request\_head文件模拟HEAD请求头

返回重新封装的请求信息

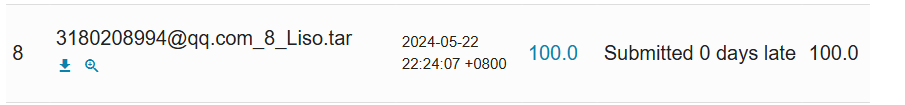


使用request\_hahe文件模拟HAHE请求头

返回"HTTP/1.1 501 Not Implemented\r\n\r\n"



自动测试平台得分100.0

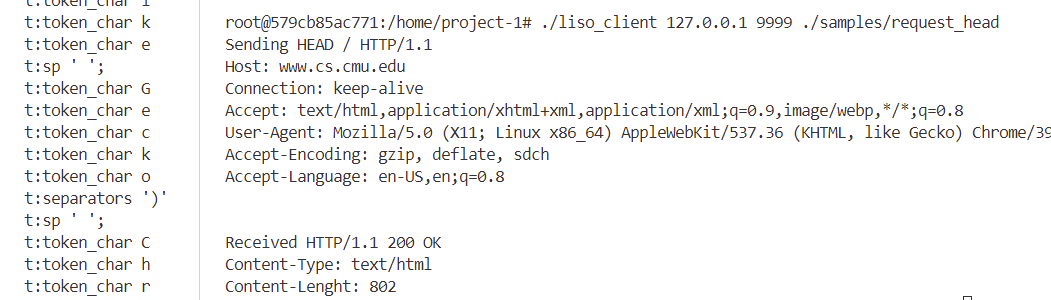


## **5.2 HEAD、GET、POST方法的实验结果与分析**

服务器对于HEAD请求的响应：

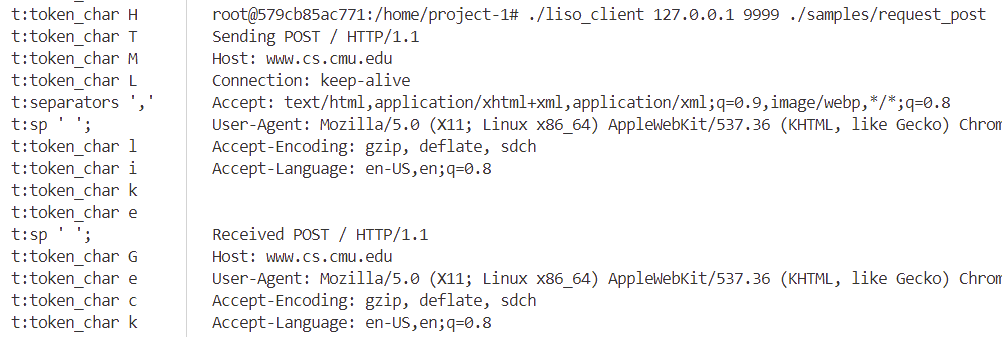
客户端收到协议状态HTTP/1，1 200 OK，Content-type资源类型和Content-length内容长度。

状态码可用于测试资源的可用性，长度可以确定文件是否值得下载。



服务器对于POST请求的响应：

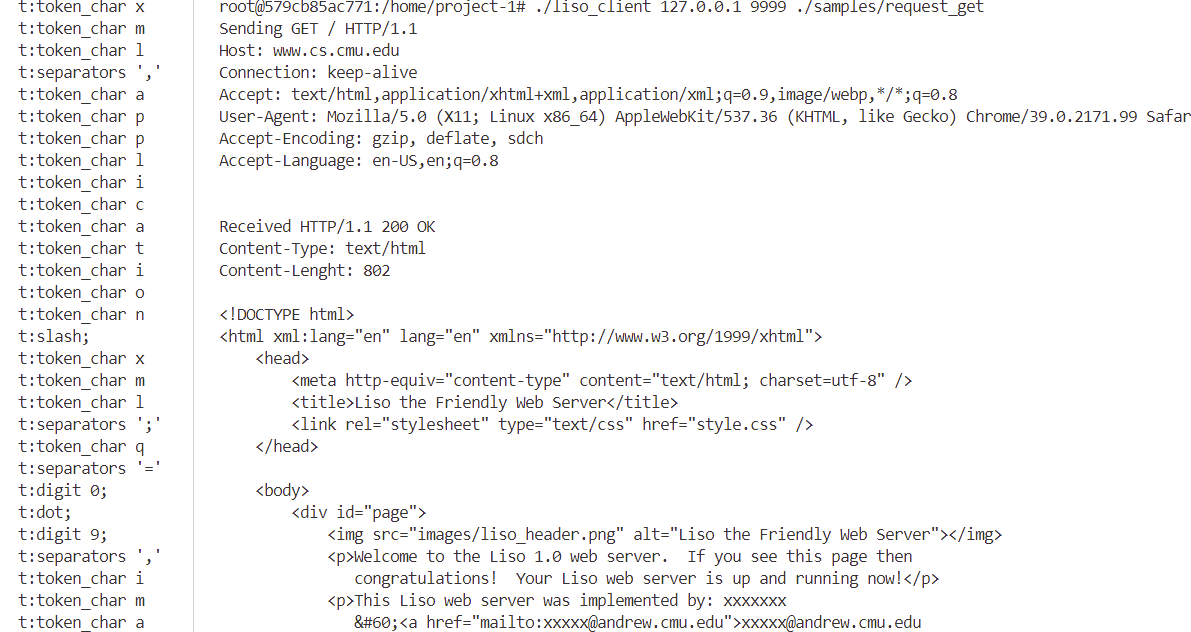
和echo返回相同，客户端直接收到它发送的原数据。



服务器对于GET请求的响应：

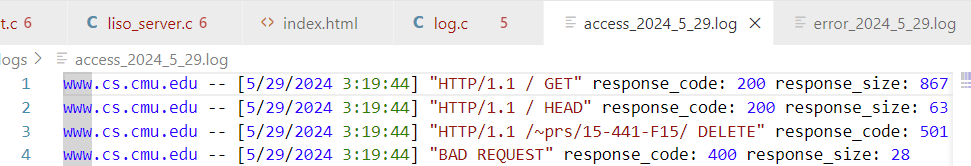
客户端收到制定资源的内容和状态码，内容类型，内容长度等。

GET只能获取指定资源的数据，不能应用于修改数据，并且该请求可被缓存，也可以保存在浏览器的历史记录中。



日志记录状态

该日志记录了网站名称，记录的日期和时间，协议类型，响应的状态码，响应的长度

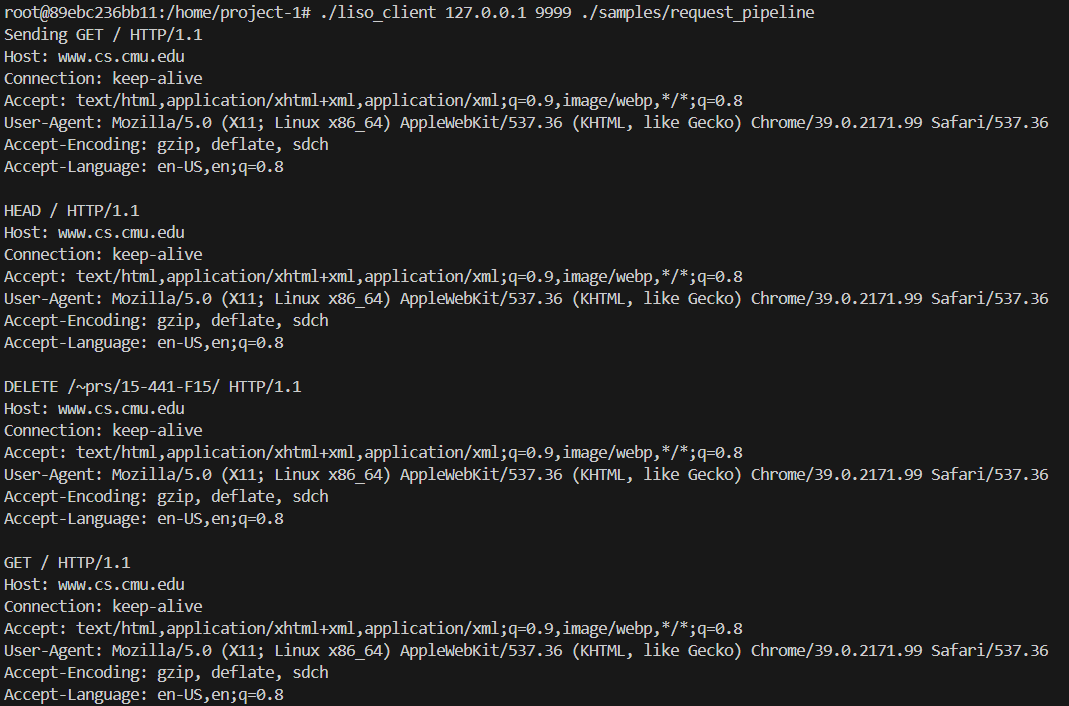


自动化测试平台成绩

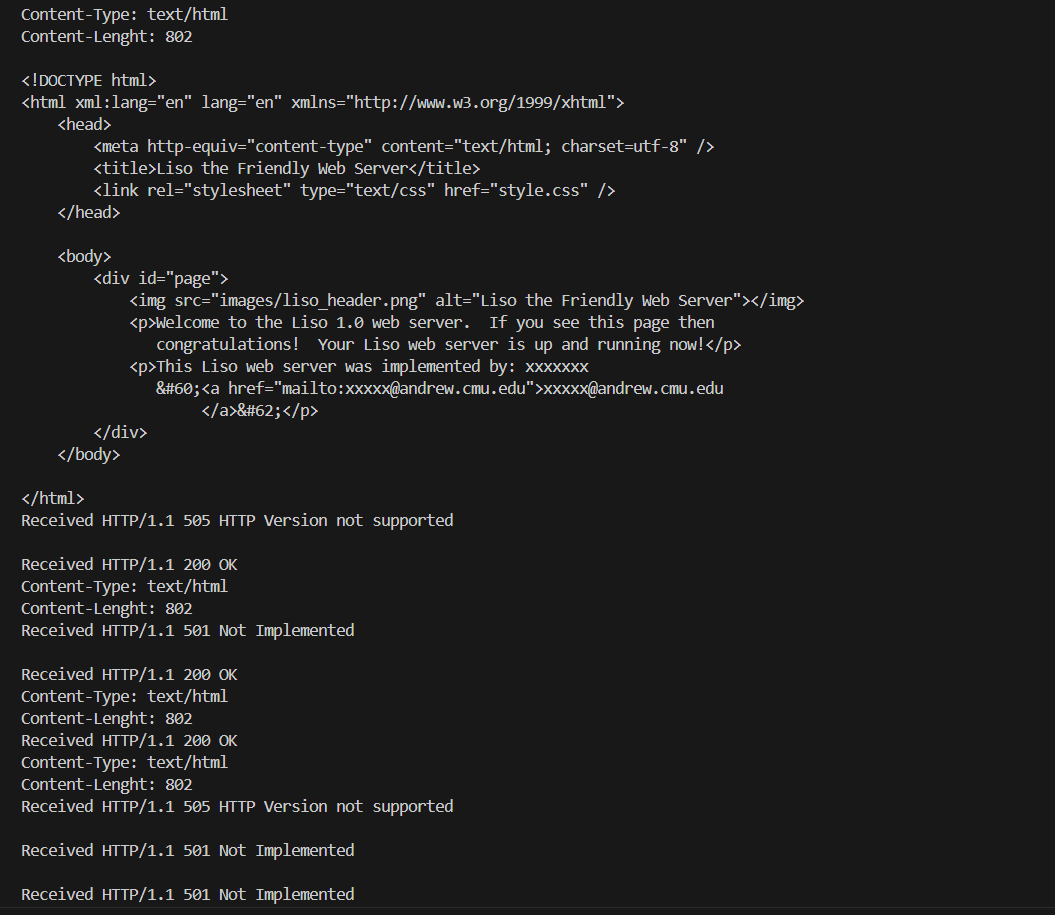


## **5.3 HTTP的并发请求的实验结果与分析**

客户端发送多个请求



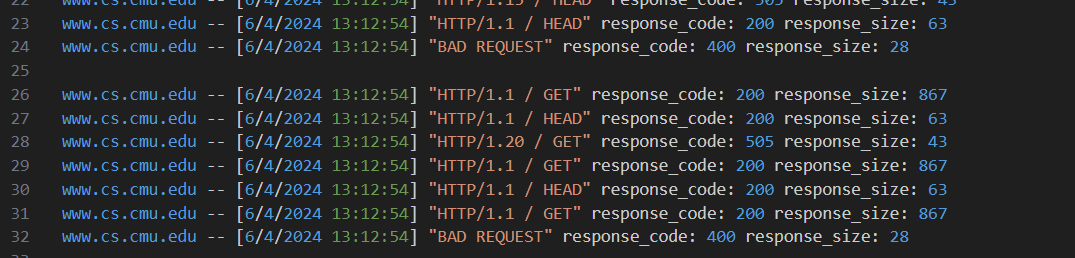
客户端接收到请求



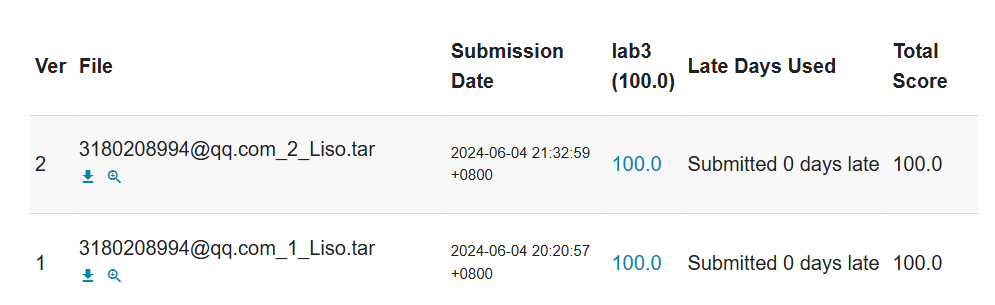
通过上图可以看出，服务器对于流水线传入的请求进行了处理，并且流水线形式返回了相应的相应信息，如 HTTP/1.1 200 OK ,HTTP/1.1 505 HTTP Version not supported,HTTP/1.1 501 Not Implemented 以及相应的html片段。说明流水线处理成功。

流水线传入请求传入了29个，任务圆满完成。

相应的生成日志:

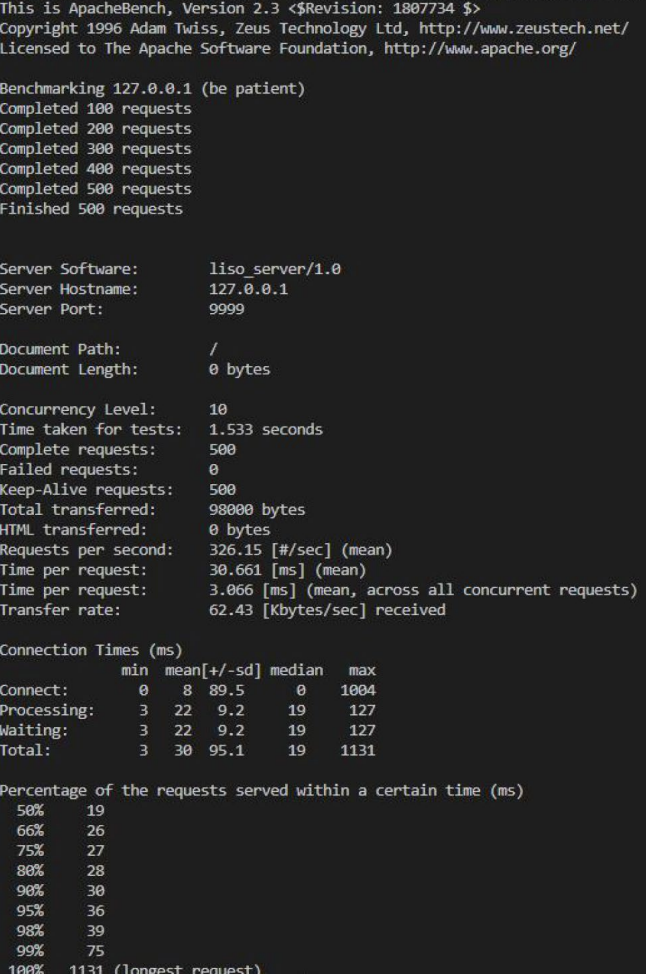


自动化测试平台得到100分

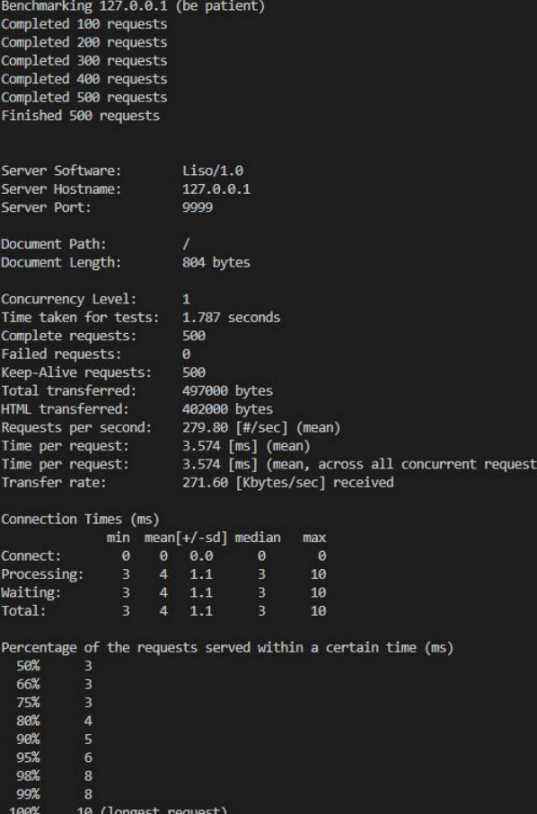


## **5.4 多个客户端的并发处理的实验结果与分析**

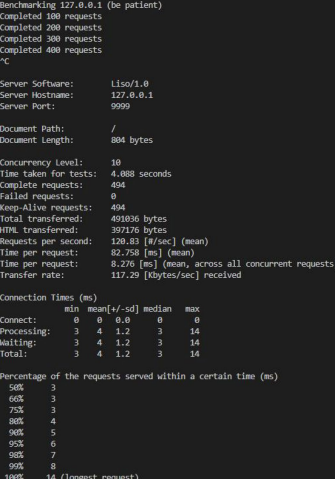
对本周进行测试：



对第三周进行测试，和1个客户端并发



和10个客户端并发

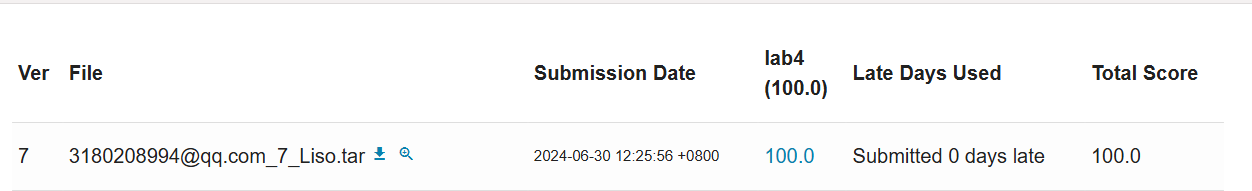


第三周在并发10个时最多并发494个请求。

由上图可知，经多次测试对比，第三周的代码在运行一部分请求之后超时发生在多个客户端的时候，当客户端个数为1时，请求能够正常进行，因此得知第三周没有正确实现客户端的并发操作。而在第四周的时候，采取多个客户端时所有的请求能够正常运行。

在没有实现并发的客户端中并发时会造成和客户端的连接发生问题从而堵塞，二实现了并发客户端的不仅效率更高，也可以减少堵塞情况。

自动化测试平台成绩



# 六、总结

总结自己在实践过程中遇到的各类问题、困难以及解决过程中的收获，对实践内容等方面的体会与建议。

Week1

第一周的任务全部完成，有关第二周的内容待定。

|  |  |  |  |
| --- | --- | --- | --- |
| **任务点** | **完成内容** | **完成情况** | **说明** |
| 1、 | 1）阅读HTTP/1.1的标准文档RFC2616 | 完成 | Socket 编程原理：  Client创建套接字。Connect()函数请求与服务器的连接  Server创建套接字，使用bind()函数将套接字与端口号进行绑定，listen()函数监听该端口号，accept()接收来自客户端的连接请求。 |
| 2）搭建编程环境，熟悉Socket编程方法； |
| 3）掌握lex和yacc正确解析消息（message）的方法 |
| 2、协议设计 | 1）详细阐述请求消息的解析方法。 | 完成 | 使用lex和yacc两种语法分析器生成器，在lex中定义语法规则将输入的文本切割成小的语法单元(token),然后使用Yacc定义从lex文件中得到的token，使用这些token来构造解析树，并完成对应的语义动作，如存进结构体等，完成对每一部分的解析。 |
| 2）说明接收缓冲区的设计； | 完成 | Client设计msg缓冲区存放读入的文件，使用buf缓冲区接收来自server的返回信息。  Server端设计部分缓冲区，接受来自client的数据。 |
| 3）说明日志记录模块的设计。 | 第二周，待定 |  |
| 4）其它设计细节。 | 完成 | 在server端调用parse解析函数，使其能够解析来自客户端的请求消息。在client端添加命令行参数的值，使输入时可以接收文件，完成多请求行的操作。 |
| 3、协议实现 | 1）能够正确解析客户端消息（带多请求行），识别出GET，HEAD，POST，并返回（echo）给客户端。 | 完成 | 使用request结构体，headers指针。通过对比request->headers与GET，POST，HEAD是否相同来进行echo判断 |
| 2）能够正确解析客户端消息，识别出不是GET，HEAD，POST的其它方法，并返回代码为501的响应消息。 | 完成 | 与识别GET等请求头原理相同，若不符合，则返回501的响应消息 |
| 3）能够正确解析客户端消息，识别出5种以上格式错误，并返回代码为400的响应消息。 | 完成 | 不同的格式错误返回相同的400 bad request |
| 4、实验结果及分析 | 1）给出所实现任务点的测试样例，并将测试结果截图展示； | 完成 | 分别使用request\_get,reguest\_post,request\_head,  request\_bad,request\_hahe五个文件来  模拟三种请求头，错误的请求头格式和不符合条件的请求头，进行测试。 |
| 2）对结果进行合理的分析说明； | 完成 | 结果说明简易echo server的设计已经成功，可以识别GET,POST,HEAD以及其他请求头，并识别错误格式。根据不同的结果返回相应的响应信息 |

**Week2**

|  |  |  |  |
| --- | --- | --- | --- |
| **任务点** | **完成内容** | **完成情况** | **说明** |
| 1、协议设计 | 1）详细阐述HTTP1.1三种基本方法的设计规则（persistent connection）和协议头部格式； | 完成 |  |
| 2）说明接收缓冲区的设计； | 完成 |  |
| 3）说明日志记录模块的设计。 | 完成 |  |
| 4）其它设计细节。 | 完成 |  |
| 2、协议实现 | 1）实现HTTP1.1三种基本方法（persistent connection）； | 完成 |  |
| 2）实现接收缓冲区； | 完成 |  |
| 3）实现日志记录模块； | 完成 |  |
| 4）实现读写磁盘文件时遇到的错误的处理 | 完成 |  |
| 5）其它实现细节。 | 完成 |  |
| 3、实验结果及分析 | 1）给出所实现任务点的测试样例，并将测试结果截图展示； | 完成 |  |
| 2）对结果进行合理的分析说明； | 完成 |  |
| 3） 实验结果包含正确的日志格式记录运行过程。 | 完成 |  |

**Week3**

|  |  |  |  |
| --- | --- | --- | --- |
| **任务点** | **完成内容** | **完成情况** | **说明** |
| 1、协议设计 | 详细阐述HTTP pipelining的设计。 | 完成 |  |
| 2、协议实现 | 实现HTTP pipelining。 | 完成 |  |
| 3、实验结果及分析 | 1）给出所实现任务点的测试样例，并将测试结果截图展示； | 完成 |  |
| 2）对结果进行合理的分析说明； | 完成 |  |
| 4、 | 给出能够正确解析的并发请求最大数量 | 完成 |  |

Week4

|  |  |  |  |
| --- | --- | --- | --- |
| **任务点** | **完成内容** | **完成情况** | **说明** |
| 1、协议设计 | 详细阐述并发客户端的设计方法与技术。 | 完成 |  |
| 2、协议实现 | 实现并发客户端。 | 完成 |  |
| 3、实验结果及分析 | 1）实验结果及分析； | 完成 |  |
| 2）并发效果以及性能测试；  使用Apache bench，设置不同数量级的并发数量，对第三周实现的服务器（不支持并发客户端）和第四周实现的服务器（支持并发客户端），进行对比测试，分析测试结果中的各项性能参数，并讨论本周工作对服务器性能提升所起到的作用。 | 完成 |  |
| 4、 | 能够支持的并发客户端的最大数量 | 完成 |  |