**《计算机网络》课程设计报告**





**HTTP 的设计与实现**

**学 号**

**姓 名**

**学 院 求是学部**

**专 业 智能机器平台**

**年 级 2019级**

**任课教师 赵增华**

**2021 年 9 月 29 日**

1. **报告摘要**

本项目主要内容为使用BSD Socket API（Berkeley Sockets API）实现具有并发性的web server。依据 RFC 2616文档，实现 HTTP/1.1的HEAD, GET, POST 等基本功能，并支持并行请求和并发接入等功能。

在第一周的实践中，我们详细阅读了HTTP/1.1的标准文档 RFC 2616，学习了Linux系统的相关指令与操作，初步学习了Socket编程方法，并基于需求与所学知识对lex和yacc进行修改，最终实现了简单的echo web server，能够实现预期功能。

在第二周的实践中，我们进一步完善了我们的echo\_server，使其能够解析GET、HEAD、POST三种方法，并能实现400，404，408，501，505五种错误，并且创建了标准格式的日志模块。

在第三周的实践中，我们进一步修改parse文件，创建规则建立新的结构体，使得服务器能够按顺序正确解析较高数量的并发请求。

在第四周的实践中，我们通过完善server文件，使得服务器在等待一个客户端发送下一个请求时，能够同时处理来自其它客户端的请求，即能够同时处理多个并发的客户端。

本报告主要包含需求分析、协议设计文档、协议实现、实验结果及分析几个部分。

1. **需求分析**

HTTP协议是客户端和服务器端两者通信共同遵循的一些规则。主要内容是定义了客户端向服务器请求资源，服务器响应客户端请求的规则与形式。依据不同功能模块进行需求分析如下：

**2.1 方法实现模块**

HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接。

当一个网页文件中包含了很多图像的地址的时候，那就需要很多次的http请求和响应，每次请求和响应都需要一个单独的连接，每次连接只是传输一个文档和图像，上一次和下一次请求完全分离。即使图像文件都很小，但是客户端和服务器端每次建立和关闭连接却是一个相对比较费时的过程，并且会严重影响客户机和服务器的性能。

HTTP 1.1应能够支持各类方法的持久连接（HTTP/1.1的默认模式应使用带流水线Pipelining的持久连接），在一个TCP连接上可以传送多个HTTP请求和响应，尽可能地减少建立和关闭连接的消耗和延迟。比如一个包含有许多图像的网页文件的多个请求和应答应可以在一个连接中传输。

HTTP1.1还应支持HTTP/1.0所不能实现的文件断点续传。使得在下载或上传时，将下载或上传任务（一个文件或一个压缩包）人为的划分为几个部分，如果碰到网络故障，可以从已经上传或下载的部分开始继续上传下载未完成的部分，而没有必要从头开始上传下载。此功能可以使用户节省时间，提高速度。

**2.2 错误检测模块**

HTTP 1.1中应对现有的错误状态进行完善，并增加更多的错误状态响应码，如请求的资源与资源的当前状态发生冲突、服务器上的某个资源被永久性的删除等目前不支持的错误状态。更多的错误状态可以更加灵活地反映当下情况，给与访问者与开发者更加精准的反馈。

**2.3 日志模块**

HTTP 1.1应支持规范化的日志记录，能够按照Apache日志文件中“Error Log”的格式记录服务器的出错情况，按照 Apache 日志文件中“Access Log”的“Common Log Format”记录服务器处理的请求。

日志应有较强的可读性和区分性，详细记录各类情况以及发生环境，以便日后查阅与调试。

**2.4 pipelining模块**

在网络服务中，用户希望服务器能够支持客户端连续发送请求，并能按照发出请求的顺序依次接收到服务器的响应，即支持HTTP pipelining。

HTTP 1.1应能允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，服务器端则按照接收到客户端请求的先后顺序依次回送响应结果，并能保证客户端能够区分出每次请求的响应内容，尽可能地减少整个下载过程所需要的时间。并且pipelining模块应与错误检测模块进行配合，对于 HTTP 的并发请求，如果服务器认为其中一个请求是错误的并拒绝该请求，那么服务器需要能够正确识别并解析出并发到达的下一条请求。

在HTTP 1.1中，长链接应当能够得到良好的控制，以确保pipelining能够在可控可调节的情况下进行。request和reponse头中应出现一个控制长链接的头，其含义是client和server通信时对于长链接的处理方式，并能提供更多的选择性。

在HTTP 1.1中，client和server都应默认对方支持长链接，但若client使用http1.1协议，但又不希望使用长链接，只需在header中指明该请求头的值为一特定值即可完成此操作；如果server方也不想支持长链接，则同样只需在response中对该值进行指定。

二者都表明当前正在使用的TCP链接在当天请求处理完毕后会被断掉，此后client再进行新的请求时必须创建新的TCP链接。

**2.5 并发处理模块**

HTTP1.1应支持当服务器在等待一个客户端发送下一个请求时，能够同时处理来自其它客户端的请求，使服务器能够同时处理多个并发的客户端。

在这个过程中应允许突发事件的发生，如客户端可能会“暂停”（即请求发送了一半突然暂停）或出错，但HTTP1.1协议应保证这些问题不对其他并发用户产生不良影响。也就是说，如果一个客户端只发送了请求的一半就停止了，那么服务端应能够继续为另一个客户端提供服务。

**2.6 其他性能**

HTTP 1.1在继承了HTTP 1.0优点的基础上，应通过增加更多的请求头和响应头来改进和扩充HTTP 1.0的功能，克服HTTP 1.0的一些性能问题。

**1. 对于虚拟WEB站点和HOST头的管理**

HTTP 1.0不支持Host请求头字段，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。WEB浏览器无法使用主机头名来明确表示要访问服务器上的哪个WEB站点，这样就无法使用WEB服务器在同一个IP地址和端口号上配置多个虚拟WEB站点。

在HTTP 1.1中，WEB浏览器应能够使用主机头名来明确表示要访问服务器上的哪个WEB站点，实现在一台WEB服务器上可以在同一个IP地址和端口号上使用不同的主机名来创建多个虚拟WEB站点。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。

**2. 对于身份与状态的管理**

HTTP 1.1还需提供与身份认证、状态管理相关的请求头和响应头。这些响应头用于身份的验证，与状态的管理与反馈，保证连接的安全性与可靠性，同时便于实时监控状态。

**3. 对于缓存的处理**

在HTTP 1.0中主要使用header里的Expires，If-Modified-Since来做为缓存判断的标准，HTTP1.1应引入更多可供选择的缓存头来控制缓存策略，通过不同发的缓存策略在不同情况下更好的管理和利用缓存，避免空间的浪费，并保证数据传输和方法实现（请求与响应）的速度。

**4. 对于带宽优化及网络连接的使用**

HTTP1.1应当克服一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了。为解决这个问题，HTTP1.1应能支持断点续传，它应在请求头引入range头域，允许只请求资源的某个部分，方便开发者自由选择以便于充分利用带宽和连接。

1. **协议设计**

**3.1 总体设计**

基于协议需要实现的功能，将协议划分为方法实现模块、错误检测模块、日志模块、pipelining模块、并发处理模块。

**1. 方法实现模块**

用于实现HEAD，GET，POST方法，其中包括GET，HEAD，POST三个主要功能函数，用于调用方法的use\_method函数等。

**2. 错误检测模块**

用于检测400、404、408、501、505等错误并基于不同的错误形式给出不同的处理方案，错误检测模块中一部分（如404、500）集成在方法实现函数中，在接收和使用方法时进行错误检测并给出相应响应。而另一部分（如501）集成在方法调用函数中，用于检测在调用方法时产生的错误并给出相应的响应。

**3. 日志模块**

用于记录客户端请求和服务器端响应的历史信息，便于后期的查询与调试，该模块相对独立，其主要函数为my\_log函数，采用写入txt文件的记录方式记录格式化日志（包括错误日志和访问日志）。在方法解析与实现的各种情况时以不同的参数调用该函数，记录该情况所对应的发生时间、进程码、ip地址等信息。

**4. pipelining模块**

用于实现服务器能连续响应客户端使用同一个 TCP 连接同时发送的多个请求，同时该模块还能识别出某一请求中的错误并拒绝该请求，同时正确识别并解析出并发到达的下一条请求。

**5. 并发处理模块**

用于使服务器在等待一个客户端发送下一个请求时，能够同时处理来自其它客户端的请求。除此之外，在客户端“暂停”或出错时，使服务端能够继续为另一个客户端正常提供服务。该模块主要内容为select()方法的应用，作用是实现多用户并发连接。

在实现协议功能时这五个模块协同作用，相互调用，在不同情况下实现不同的功能，共同实现协议的目的。

**3.2 数据结构设计**

我们通过定义协议头部结构和不同的结构体来实现客户端的功能。

协议的头部结构如下图所示：request请求由三部分组成。第一部分为request\_line，该部分由一行元素组成。第二部分为request\_header，该部分可以包括0或多行内容，每一行称之为request\_header\_part。最后由一个crlf结尾。

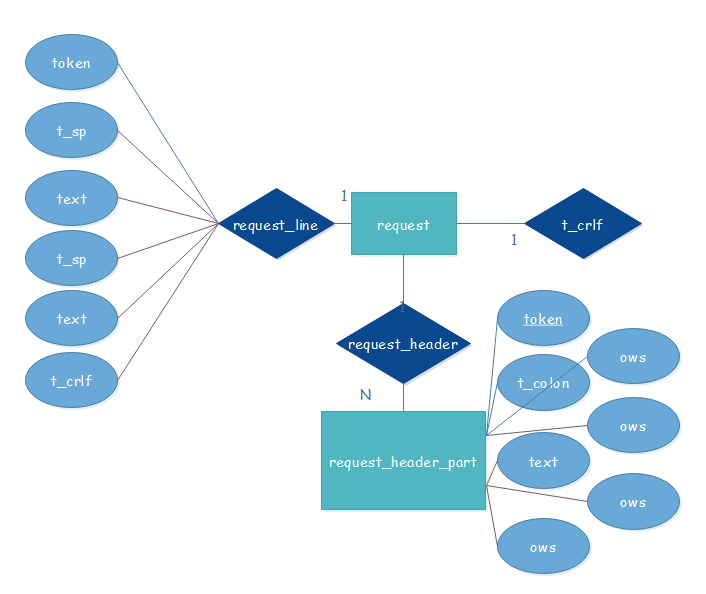


图3-1 request结构分析

服务器程序中所定义客户端地址结构体如下图：

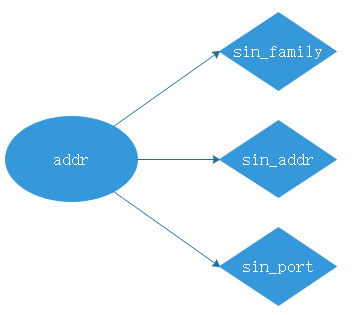


图3-2 addr结构体

该结构体中含有三个元素：分别是代表接入端口的sin\_port，代表客户端地址的sin\_addr以及代表客户端ip地址类型的sin\_familly。

客户端程序中所定义hints结构体如下图：

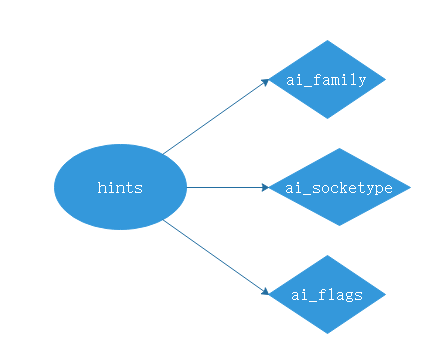


图3-3 hints结构体

该结构体中含有三个元素：分别是代表socket类型的的ai\_socketype，代表填充本机地址内容的标记ai\_flags以及代表ip地址类型的ai\_familly。

协议中定义的request结构体如下图：

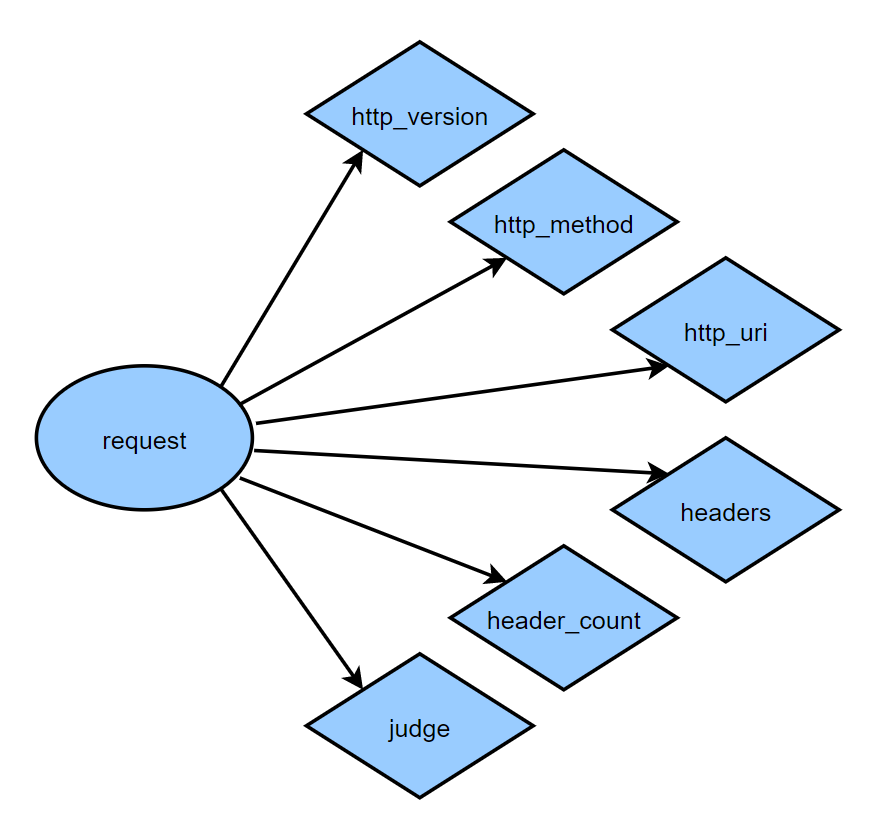


图3-4 request结构体

该结构体中含有六个元素，分别代表：表示HTTP请求方法标记的字符数组、表示HTTP版本号数组、表示Request-URI的字符数组、标记请求格式错误的judge符号、表示请求头中所含request\_header\_part数目的计数器以及指向具体头域内容的headers指针。

指向的具体头域内容的同样是一个结构体，见下图：

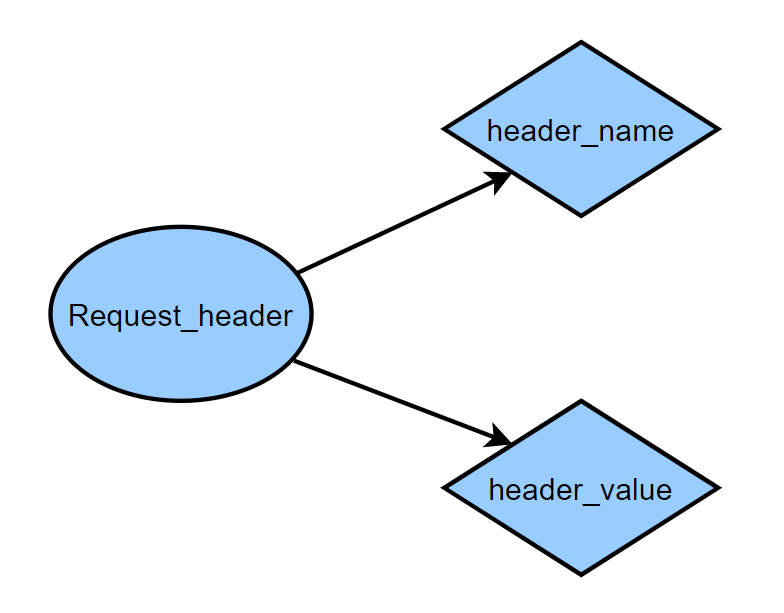


图3-5 request\_header结构体

该结构体包含两个元素，分别是请求头的名字与内容。

**3.3 协议规则设计**

基于需求与目的，我们制定的协议实现规则与机制如下：

**方法实现**

客户端向服务器发送请求，该请求可以包含GET，HEAD，POST方法，服务器中使用三个同名功能函数GET，HEAD，POST来实现客户端请求中的方法。

在检测到客户端发来的请求时，服务器对该请求进行解析，对其格式进行识别，在无格式错误等各类错误的情况下，调用use\_method函数进行方法的调用，并对于各类方法返回相应的响应。

**错误响应**

服务器在接收到客户端传来的请求时，调用错误检测模块对其进行实时检测，在识别出请求格式错误等其他错误时，转到错误处理过程，返回错误信息。

**日志记录**

无论是正常访问实现方法还是发生错误，服务器都将调用日志模块对情况进行记录。

**HTTP Pipelining**

此外，协议支持同一客户端的并发请求，即支持HTTP pipelining，在同一客户端在同一TCP连接中同时发送的多个请求GET、HEAD、POST时，则服务器端则会调用pipelining模块，按照顺序处理 HTTP 的并发请求。

在此过程中服务器将调用错误检测模块对每个请求及其出现情况进行错误检测，若其中一个请求是错误的服务器则拒绝该请求，正常识别并解析并发到达的下一条请求。

**多客户端并发**

另外，服务器还支持多个客户端的并发处理，调用并发处理模块使得服务器在等待一个客户端发送下一个请求时，同时处理来自其它客户端的请求。

服务器在这个过程中再调用错误检测模块，如果客户端在请求过程中“暂停”或出错，该模块将检测出这些问题，并予以处理，在不对其他并发用户产生不良影响的情况下，继续为另一个客户端提供服务。

**3.3.1协议参数**

HTTP版本：HTTP使用一个“<major>.<minor>”数字模式来指明协议的版本号，HTTP消息的版本在HTTP-Version域被指明，HTTP-Verion域在消息的第一行中。

HTTP-Version = “HTTP”“/”1\*DIGHT“.”1\*DIHIT

URI（通用资源标识符）：通用标识符（URI）使格式化的字符串，通过名称，位置，或其他特征来识别一个资源。

http\_URL = “http:”“//”host[“:”port][abs\_path[“?”query]]

如果端口为空或未给出，就假定为80。意味着：已识别的资源存放于正在监听tcp连接的那个端口的服务器上，并且请求资源的Request-UR为绝对路径。如果绝对地址没有出现在URL里，那么应该给出“/”。

**3.3.2 HTTP消息**

**消息头（Message Headers）**

每一个头域由一个名字（域名）跟随一个“：”和域值构成。域名是大小写不敏感的。域值前面可能有任意数量的LWS。SP是首选的。头域能被延伸多行，这通过在这些行前面加一些SP或HT。

message-header = field-name“:”[field-value]

field-name = token

field-value = \*(field-content|LWS)

field-content = <the OCTETs making up the field-value and consisting of either \*TEXT or combinations of token,separators, and quoted-string>

常用头域：只能应用于传输消息。

Gereral-header = Cache-Control;

|Connection;

|Date;

|Pragma;

|Trailer;

|Transfer-Encoding;

|Upgrade;

|Via;

|Warning;

**请求(Request)**

一个请求消息是从客户端到服务器的，在消息首行里包括方法，资源指示符，协议版本。

Request = Request-Line;

\*(( general-header;

| request-header;

| entity header ) CRLF) ;

CRLF

[message-body];

请求行（Request-Line）：请求行是以一个方法标记开始，后面跟随Request-URI和HTTP-Version),最后以CRLF结束。元素是以SP字符分隔。除了最后的CRLF，CR或LF是不被允许的。

Request-Line = Method SP Request-URL SP HTTP-Version CRLF

方法（Method）：方法指明了在被Request-URI指定的资源上执行的方法。这种方法是大小写敏感的。

Method = “GET”|“HEAD”|“POST”

请求URL（Request-URI）:是一种通用资源标识符，并且它用于指定请求的请求资源。

Request-URI = “\*”|absoluteURI|abs\_path

星号“\*”意味着请求不能应用于一个特定的资源，只能应用于服务器本身，并且只能在方法不应用于一个资源的时候才被允许。Request-URI大多数情况被用于指定一个源服务器或网关上的资源。这种情况下，URI的绝对路径作为Request-URI被传输，并且URI网络位置必须在Host头域里指出。

请求头域（Request Header Fields）：请求头域允许客户端传递请求的附加信息和客户端自己的附加信息给服务器。这些头域作为请求的修饰，这和程序语言方法调用的参数语义是等价的。

Request-header= Accept|Accept-Charset|Accept-Encoding|Accept-Language|Authorization|Expect|From|Host|if-Match|if-Modified-Since|if-Range|if-Unmodified-Since|Max-Forwards|Proxy-Authorization|Range|Referer|TE|User-Agent

GET:GET方法意思是获取被请求URI（Request-URI）指定的信息（以实体的格式）。如果请求URI涉及到一个数据生成过程，那么这个过程生成的数据应该被作为实体在响应中返回而不是过程的源文本，除非源文本恰好是过程的输出。

HEAD:HEAD方法和GET方法一致，除了服务器不能在响应里返回消息主体。HEAD请求响应里HTTP头域里的头域信息应该和GET请求响应里的头域信息一致。

POST:请求服务器接受所指定的文档作为对所标识的URI的新的从属实体。

**响应(Response)**

接收和解析一个请求信息后，服务器发出一个HTTP响应消息。

response = Status-Line;

\*(( general-header)

| response-header;

| entity-header ) CRLF) ;

CRLF

[message-body];

状态行（stauts-Line）：响应消息的第一行是状态行，由协议版本以及数字状态码和相关的文本短语组成，各部分间用空格符隔开，除了最后的CRLF序列，中间不允许有CR或LF。

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

状态码（Status Code）：是一个试图理解和满足请求的三位数字整数码。

Status-Code = “501”|“400”

响应头域（Reponse Header Fields）：响应头域传送响应的附加信息，这些信息不能放在Status-Line里，这些头域给出有关服务器的信息以及请求Request-URI指定资源的更进一步访问信息。

Response-header= Accept-ranges|Age|Etag|Location|Proxy-Auteniticate|Retry-After|Server|Vary|WWW-Authenticate

1. **协议实现**

**第一周：**

本周要求：撰写 HTTP/1.1 设计文档、搭建编程环境、正确解析收到的客户端的消息、实现简单的echo web server。

基于本周要求，协议的实现主要由服务器程序以及语法解析器实现，辅以客户端程序对协议实现内容的正误进行验证。

1. 服务器与客户端的通信

服务器实现与客户端的通信主要依赖于socket所提供的函数。具体流程如下图所示：

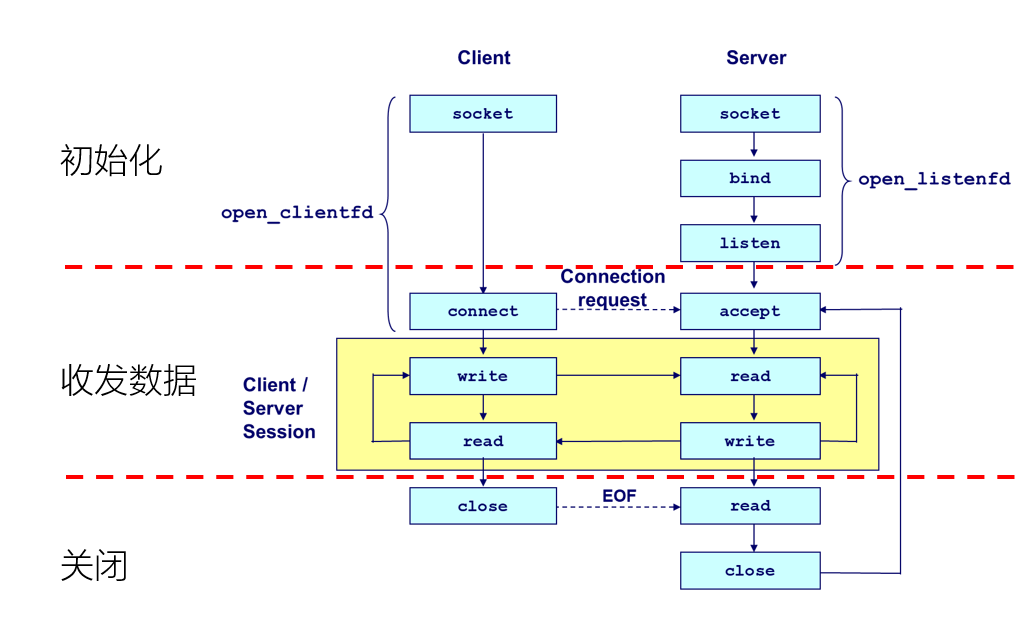


图4-1-1 服务器与客户端的通信

对于服务器来说，它的初始化包括三个过程：需要首先调用socket()函数来创建，然后调用bind()函数，将本机的地址与socket()返回值参数指定的套接字关联，最后调用listen()函数来进行监听。客户端的初始化操作只需要调用socket()函数，而不需要绑定和监听。

接下来服务端被动等待连接，而客户端主动发起连接。客户端通过调用connect()函数来发送连接请求，当客户端接收到请求后通过调用accept()函数来建立新的socket对象，用来在新建的链接上发送和接收数据。之后，客户端与服务器分别调用send()与recv()来进行数据的收发。

2. 客户端发送数据

考虑到客户端程序在终端输入数据时会因为输入的回车换行来终止输入，在此采用打开文件的方式进行相应内容的输入。

Fd\_in为文件句柄，表示文件的位置，并增加判断语句，如果fd\_in小于0代表打开失败，客户端程序返回错误信息并退出。如果正确打开：

通过上述语句将文件内容读入buf字符数组中，之后通过send()函数发送。

3. 语法解析器

通过定义语法来对请求的内容进行正确的解析。其中请求头由下述语句循环定义得到：

4. 服务端解析数据

服务端调用定义的parse()函数进行解析，并增加条件判断的部分，使其返回正确的报文。

**第二周：**

本周要求：可以概括为正确响应、持久连接、三种方法、五种错误、管理缓冲区和读写错误、记录日志等。

基于本周要求，对server文件进行进一步的完善与拓展，下面以GET方法的具体实现为例，介绍本协议对方法与错误的实现过程。

1. GET方法的具体实现

（1）检查版本

首先对方法版本进行检查，若版本不匹配返回501错误代码，若版本匹配则按照以下列方法正常实现：

在GET方法中主要利用了stat这样一个结构体。它是存储文件信息的一个结构体，在此我们使用它代表文件类型和存取的权限的属性st\_mode以及代表文件大小st\_size以及代表文件最后一次被修改的时间的属性st\_mtime。

（2）获取URI

我们利用解析出的request结构体获取具体的URI，如果URI是默认的‘/’，就将html文件的默认存储路径赋值给URI。

将URI所指的文件状态，复制到先前创建的stat结构体当中，同时判断是否绑定成功，如果不成功的话就返回404错误代码，若成功则正常进行。

（3）判断URI属性

接着利用S\_ISDIR内置函数来判断URI是否是一个文件夹，如果是文件夹，则需要将该文件夹里的可传输文件全发送给客户端。否则只需要传输用户指定的文件。

在此，自定义了exist函数用于判断用户指定的文件是否存在，只需以只读方式尝试打开文件并判断返回的文件句柄即可。

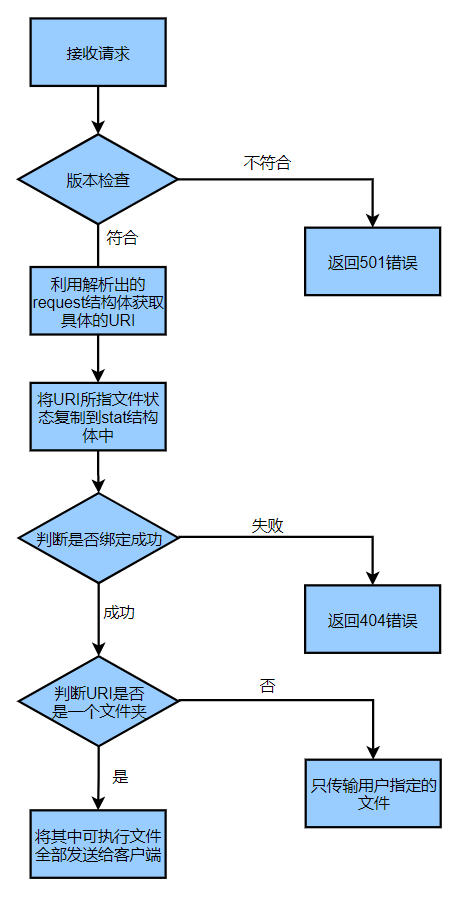


图4-2-1 GET方法实现流程图

（4）准备响应头域

最后，我们需要准备response应该具有的某些头域内容。其中主要的几个部分是Last-Modified，Content-length，Date以及Content-Type。为了正确且简洁的判断文件类型，我们对常用的文件类型进行了预处理。建立了一个哈希表用于查找，该哈希表采用直接定址法，直接通过文件类型的首字母来定位。对于Content-length可以通过st\_size获取其以字节表示的int类型结果，我们需要对其转化成字符串类型，以使其能够正确的传输。为此定义了Int2String函数。而剩下的两个部分主要是对事件的转化。这一内容需要利用time.h头文件中的函数，通过srtftime函数实现时间的格式化输出。

HEAD方法与GET方法类似，只不过不用对具体文件的内容进行读取以及发送。而POST方法由于尚未实现GCI，因此只进行简单的echo处理。

2. 错误实现

（1）400错误代码实现

通过在parser.y中定义错误的格式，对客户端的请求的格式进行识别，若为错误格式且为已在parser.y中定义的错误格式，则返回400错误。

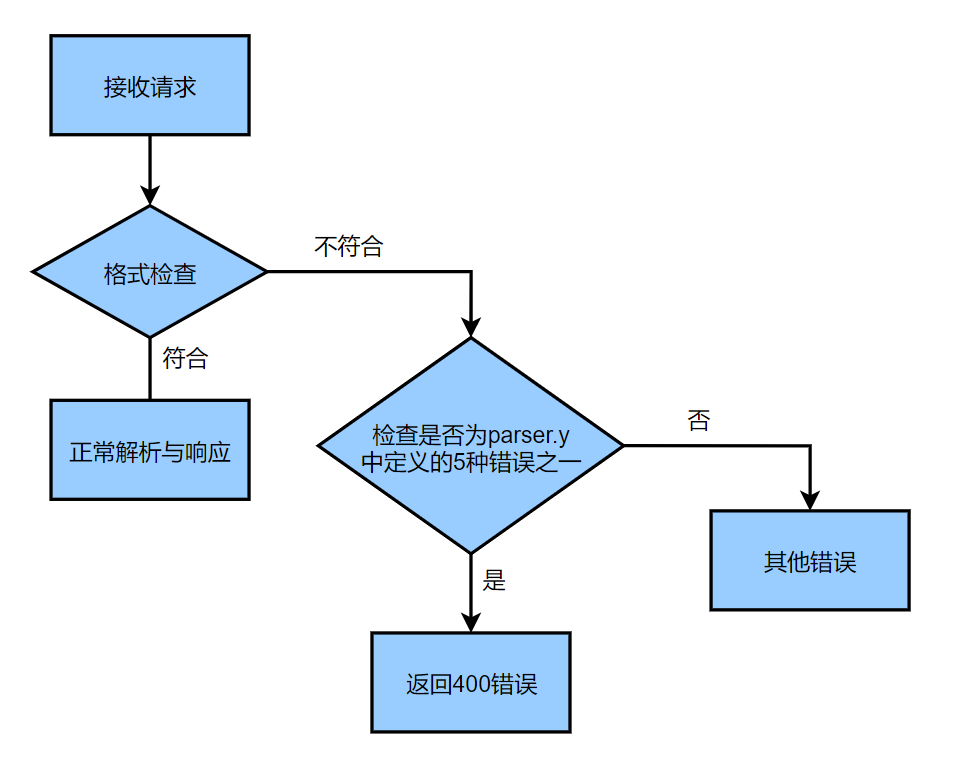


图4-2-2 400错误实现流程图

（2）404错误代码实现

然后将URI所指的文件状态，复制到先前创建的stat结构体当中，同时判断是否绑定成功，如果不成功的话就返回404错误代码。其伪代码如下：

（3）408错误代码实现

需要额外说明的是408代码的实现，这个超时我们暂时定义为用户与服务器建立连接到服务器recv函数成功接收到请求指令的时间超过了我们预定的阈值。因此，通过修改client的内容，设定一定的发送延迟，用来检验程序设定是否正确。

（4）501错误代码实现

首先对方法版本进行检查，若版本匹配则正常实现相应的方法，否则返回501错误代码。其伪代码如下：

（5）505错误代码实现

505错误代码的返回主要是使用check\_http\_version函数，通过比对request请求中包含的http\_version与server所支持的http\_version，如果两者有任何地方不同，就返回错误代码。

3. 日志模块

以创建函数的形式建立了一个简化的日志记录模块，记录格式化日志。在日志函数中采用字符串拼接的方法，在TXT文件中输出正确的日志格式。

（1）参考标准格式

首先参考了Apache 的日志文件，观察并分析了他给出的样例错误日志（Error Log）和访问日志（Access Log）的形式，结合目前服务器的功能，对于服务器实现的5种错误，在函数中分类处理，给出了5种错误情况下日志的格式。方便后期的调试和追踪。

对于服务器所能实现的GET，HEAD和POST三种方法，能够在接到不同访问请求的情况下，记录对应的访问日志，实现记录客户端操作的目的。由于访问地址为根目录，所以对应访问文件位置处是/，此处应注意。

（2）参数取得

对于错误日志，样例给出的时间格式可以直接使用ctime ()函数生成，故采用这种方式直接返回。对于访问日志，样例给的时间格式无法使用现有时间函数直接生成，所以采用strftime拼接星期、年份、时分秒等时间要素的方式来输出正确的时间格式。

对于错误日志的进程id和线程id，分别使用#include <unistd.h>和#include <sys/syscall.h> 两个语句调用getpid()和pthread\_self()两个函数所在的库，然后调用这两个函数返回进程和线程id，在错误日志中会用到。

（3）txt文件写入

使用fopen函数，”a+”状态用以创建并在末尾写入log.txt文件，方便查看与后期调试。

**第三周：**

本周要求：服务器能连续响应客户端使用同一个TCP连接同时发送的多个请求 GET、HEAD、POST，即支持 HTTP pipelining。并且服务器按照 RFC2616 规定的顺序处理 HTTP 的并发请求。另外，对于 HTTP 的并发请求，如果服务器认为其中一个请求是错误的并拒绝该请求，那么服务器需要能够正确识别并解析出并发到达的下一条请求。

基于本周要求，我们采用以下方法实现：

1. 内存排查

首先我们通过valgrind排查在请求与解析过程中可能出现的内存泄漏问题，并分析其结果，定位代码中可能出现内存管理不善的位置，对于大于3000bytes的内存空间泄露问题尤为注意，通过修改parse.c，pares.y，echo\_server等文件来将申请的内存释放掉，妥善进行内存管理。

在此过程中我们使用更加安全的函数形式以及语法来确保内存不出现泄露，如使用strncpy\_s来代替strcpy，规范使用free()函数以及malloc\_trim(0)函数。良好的内存管理可以大大增加协议的安全性和可靠性，为实现更多的并发请求打下良好的基础。

另外应注意，例如send()这类socket中自带的函数也会出现内存泄漏的情况，应注意排查。构建子串的删除函数，可以直接操作指针对内存空间进行管理，实现高效省空间的子串删除。

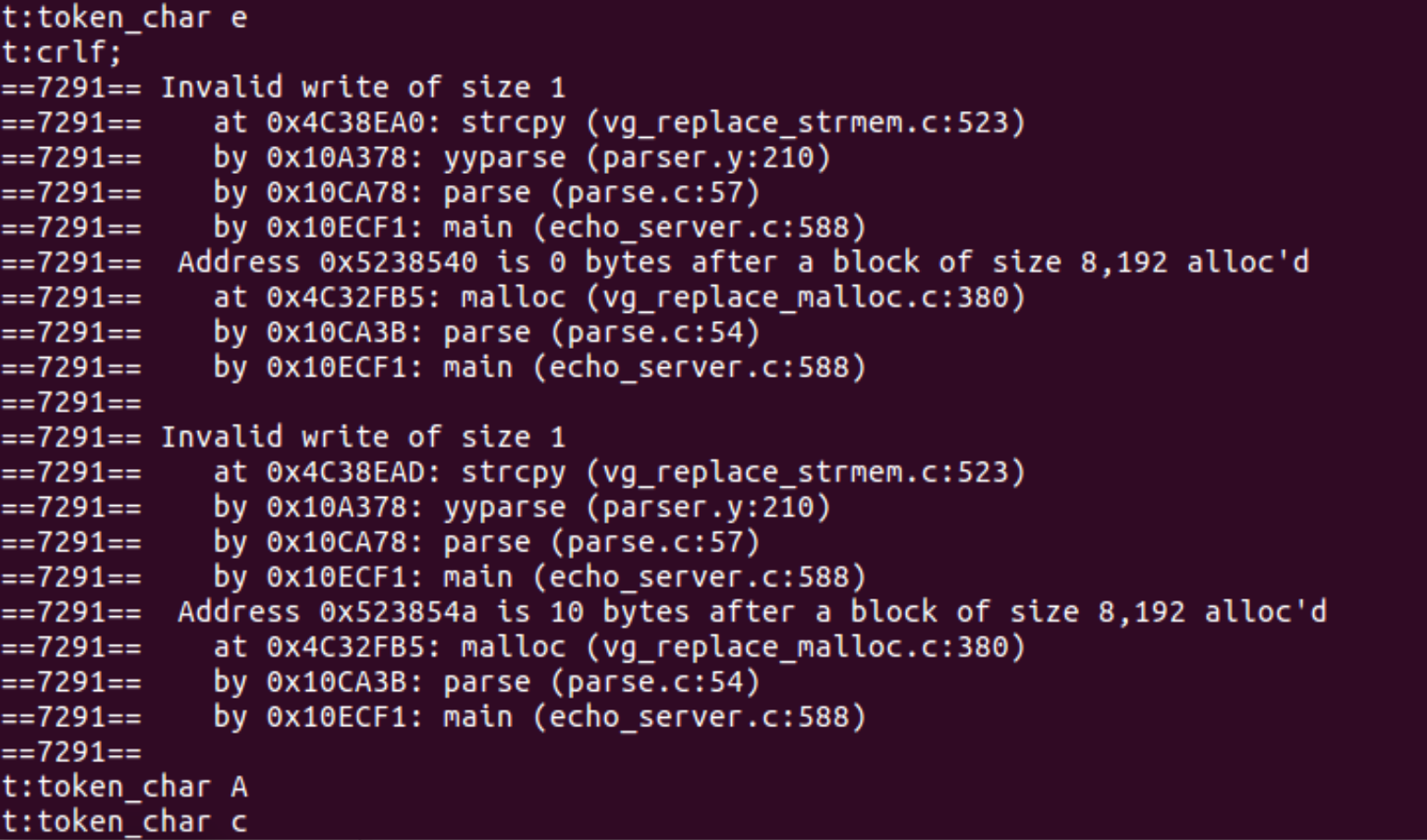


图4-3-1 valgrind内存排查过程

2. 增加规则

在修改parse文件过程中，通过增加应有规则，来读取更加长的报文格式，据此添加新的结构体来定义解析长篇报文，同时增加了一个计数器，用以记录队列中的报文数目，同时新建了功能函数，来实现更加长的报文格式。

在debug过程中产生了段错误，也通过对parse.c，pares.y等文件进行修改，增加规则使其能够实现并发请求。最终解决了段错误的问题。

3. 循环实现

后面通过一个循环来实现HTTP pipelining 并发请求，并且基于队列这一数据结构，按顺序一条条的对其进行解析，解析后的数据直接删除，再对队列中后面的数据进行解析，这样就达到了按照顺序解析Client的并发请求这一目的。

**第四周：**

本周要求：当服务器在等待一个客户端发送下一个请求时，能够同时处理来自其它客户端的请求，即服务器能够同时处理多个并发的客户端。另外，当客户端“暂停”或出错，其他并发用户不应受到不良影响，服务端应继续为另一个客户端提供服务。

基于本周要求，我们采用以下方法实现：

1. 并发准备

建立两个fd\_set类型的变量readfd与tmpfd，并通过FD\_ZERO对readfd与tmpfd进行初始化。将服务器自身的socket描述符放置在readfd中，将最大的描述符暂设定为服务器的socket。

2. 循环监听

在每次循环开始时，对tmpfd进行更新，并调用select函数。这时会将可读的文件描述符保留在tmpfd中，去掉其他的描述符。

如果可读的文件是服务器自身的socket，代表有客户端正在请求链接，通过遍历客户端列表、找到可以用来存储的位置，来存储客户端的socket。并将这个socket存入可读的readfd列表中，并更新盘符的最大值。

如果可读的文件不是服务器自身的socket，代表有客户端正在发送内容。遍历客户端列表，并依次检查该socket是否可读。在可读的基础上，接收该客户端发送的消息。

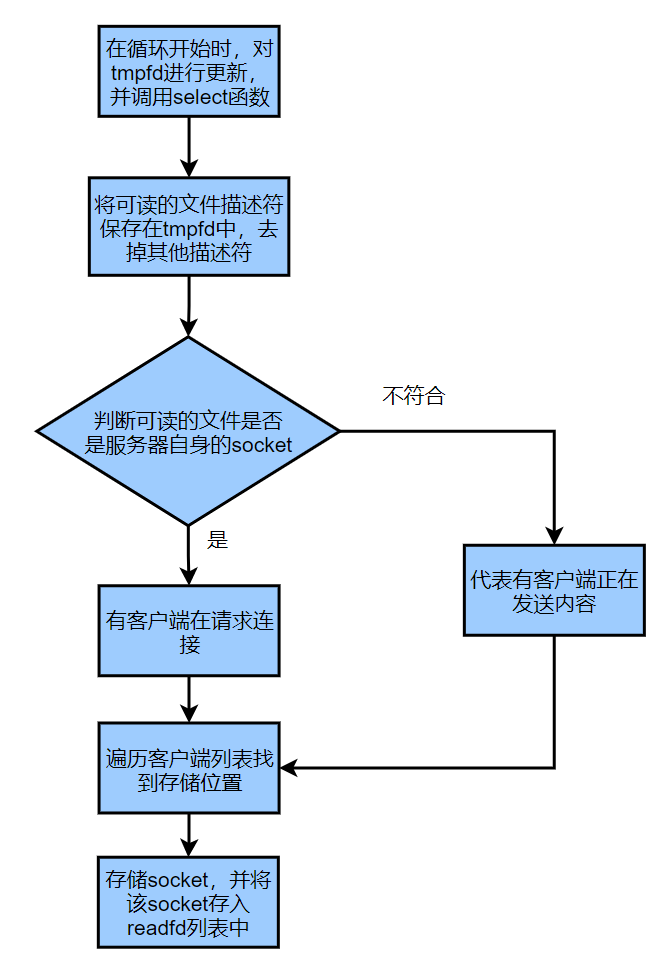


图4-4-1 循环监听示意图

3. CGI实现

这周还实现了部分CGI内容，实现了对浏览器发送至服务器内容的解析，对‘=’与‘&’两个标志符进行检测，解析出输入的表单内容。设计了HTML页面，并对相应CGI程序的编译运行过程进行了仿真实验。

完成了一个具有表单的 HTML 页面（能够实现账号和密码的输入、提交）。服务器能够根据用户在浏览器提交的信息，动态生成网页（网页显示的内容包括用户动态提交的账号和密码）

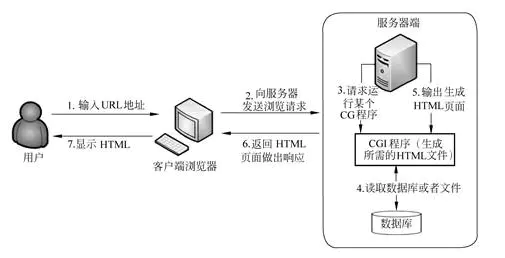


图4-4-2 CGI工作机制示意图

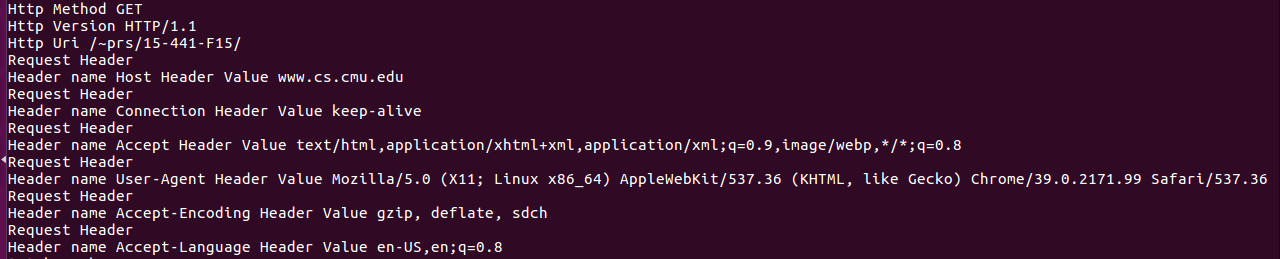
1. **实验结果及分析**

**第一周：**

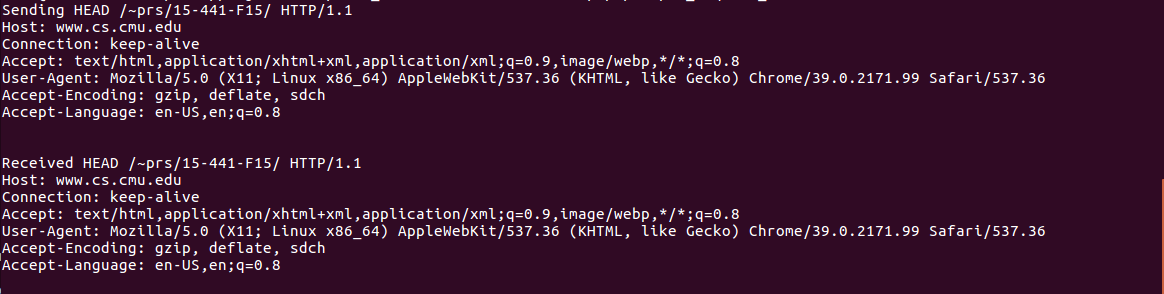
实现了简单的echo server，Server 在收到 client 的消息后，能够识别出 GET，HEAD，POST，正确解析出来，并且返回响应消息给客户端。

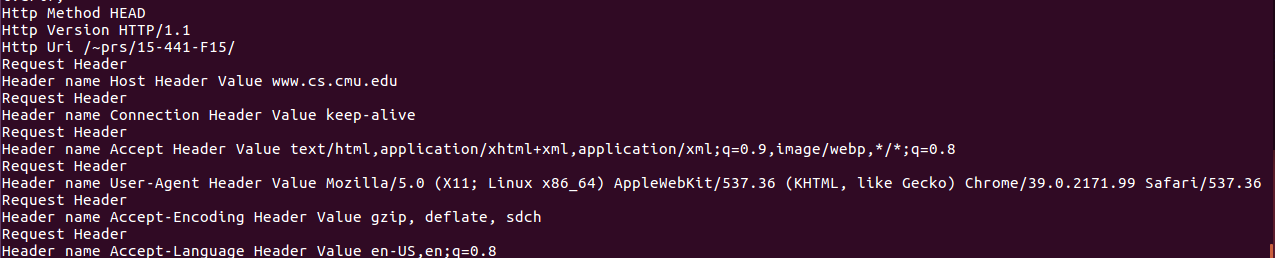
**1. GET功能**



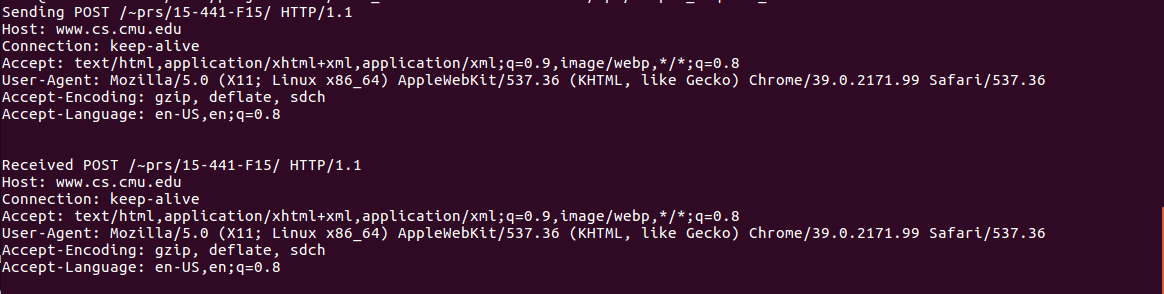


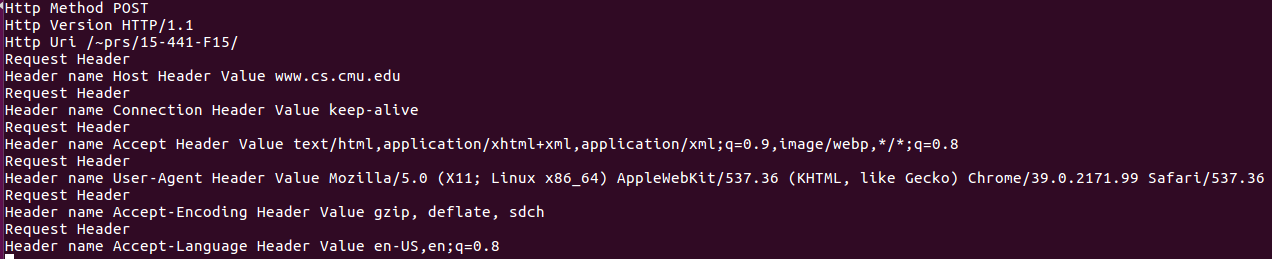
**2. HEAD功能**





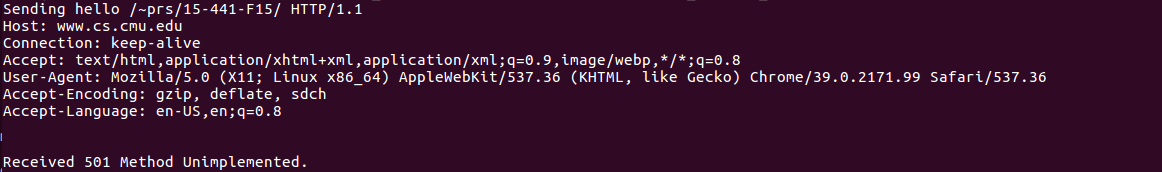
**3. POST功能**





**4. 识别出其它方法，返回501**

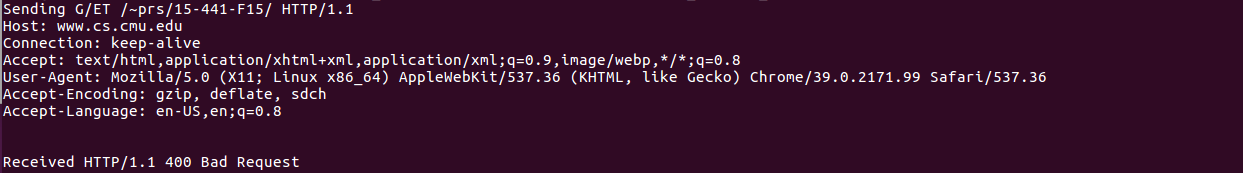
如果收到客户端发来的是除 GET, HEAD 和 POST 以外的其它方法，服务器能够识别出并不予实现，且能够返回响应消息“501 Method Unimplemented”。



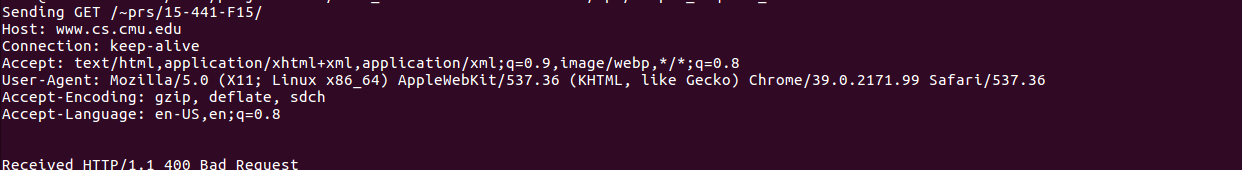
**5. 识别出 5 种格式错误，返回400**

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。

**（1）方法中含有非法字符**

****

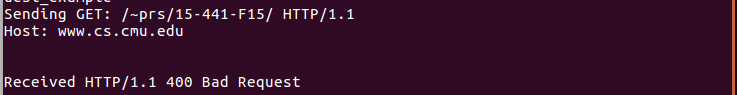
**（2）缺少版本号**



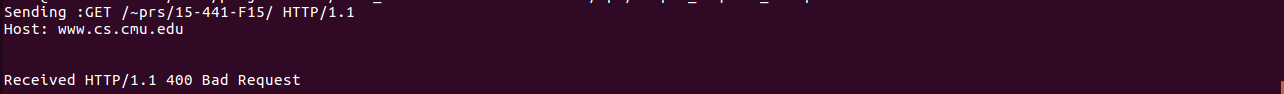
**（3）缺少方法**



**（4）方法后有多余的冒号**

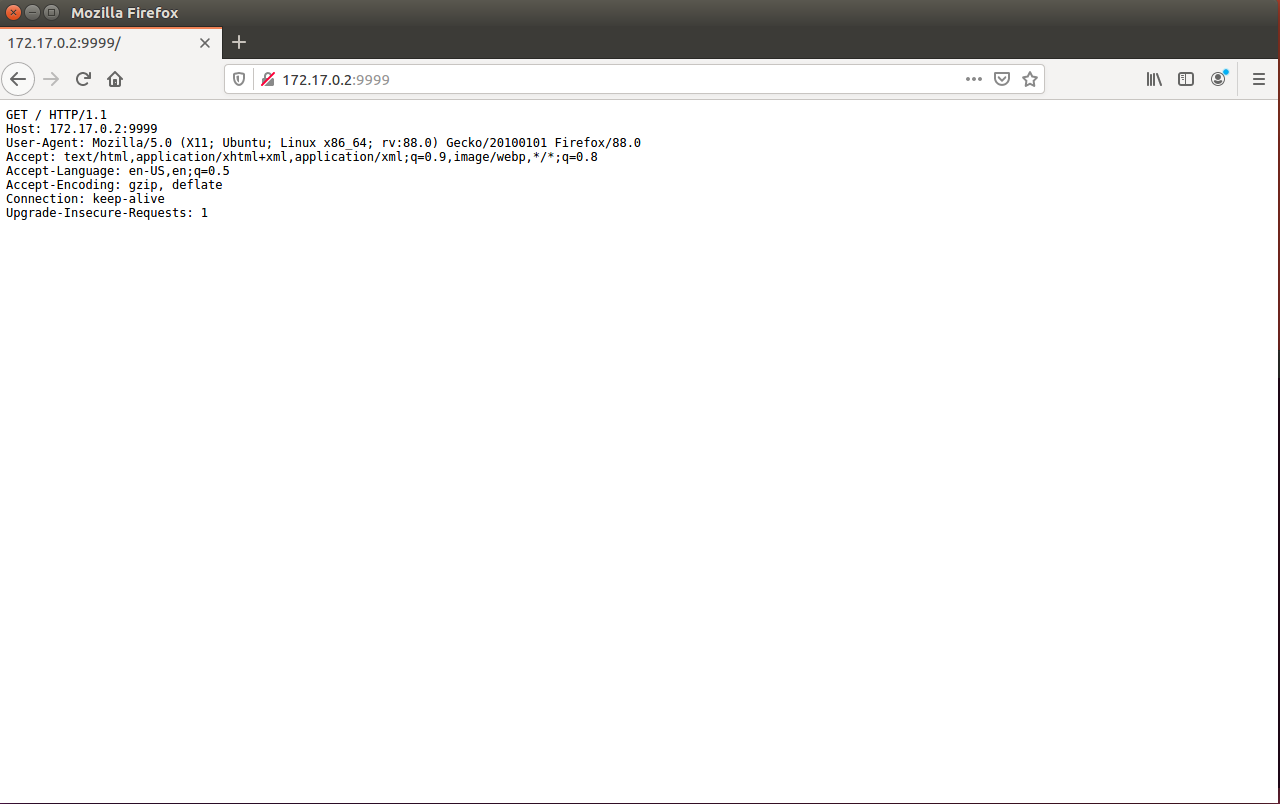


**（5）方法前有多余的冒号**



**6. 用浏览器打开**

能够用浏览器打开，实现相应功能





基于这些功能，实现了简单的echo server，Server 在收到 Client 的消息后，能够正确解析，并且根据不同情况返回响应消息，完成了第一周的任务目标。

**第二周：**

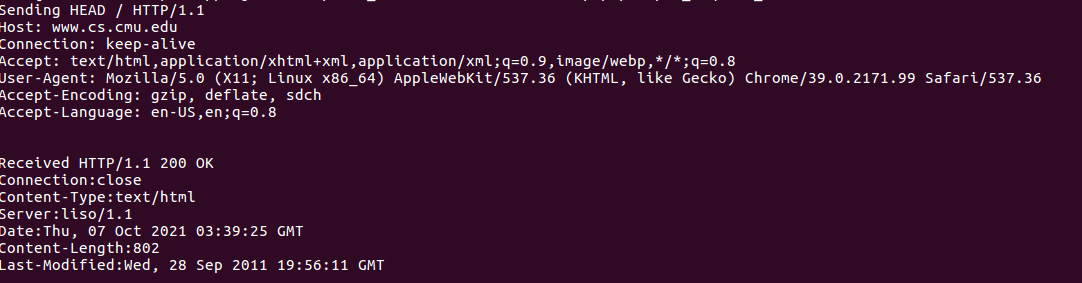
服务器成功实现了第二周的要求，能够实现正确响应、持久连接、三种方法、五种错误、管理缓冲区和读写错误、记录日志等功能。

**1.实现三种方法**

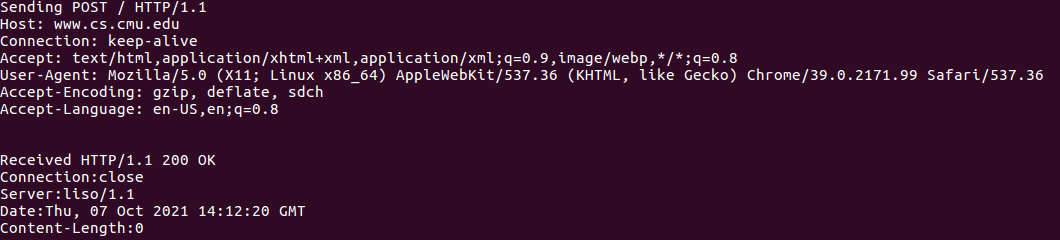
GET方法



HEAD方法

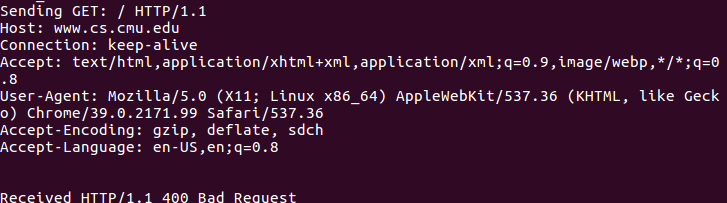


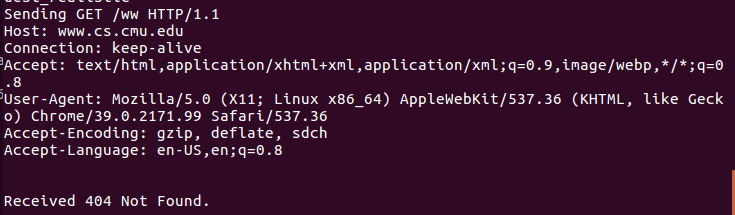
POST方法



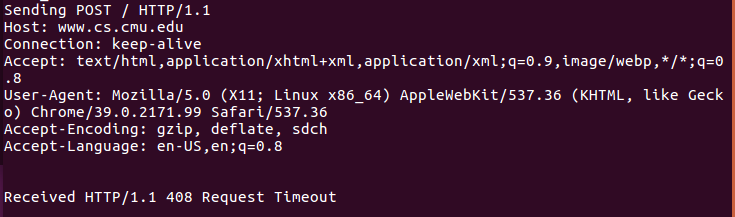
**2.支持五种错误**

400错误

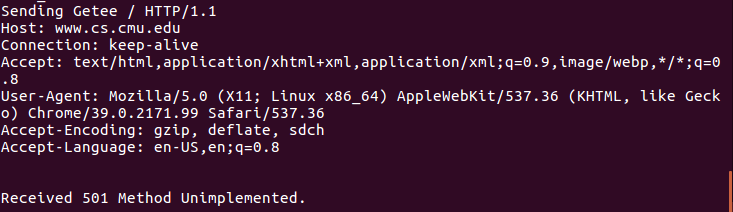
404错误



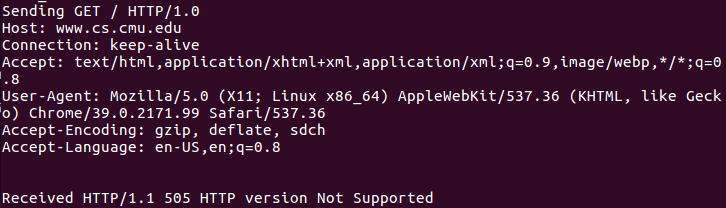
408错误



501错误

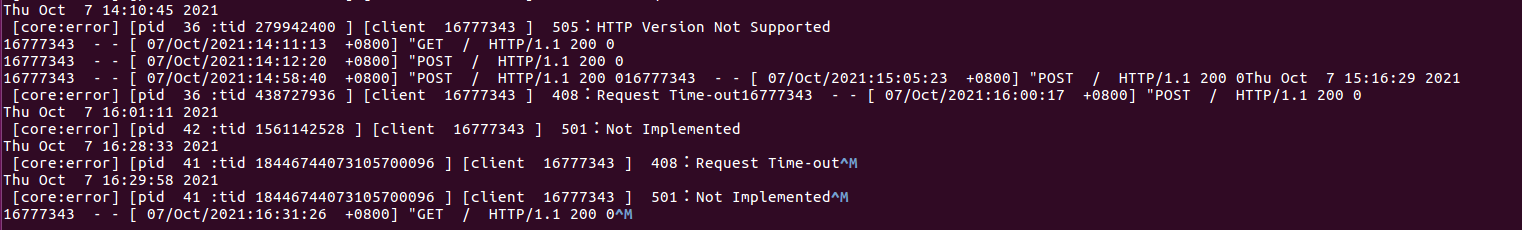


505错误

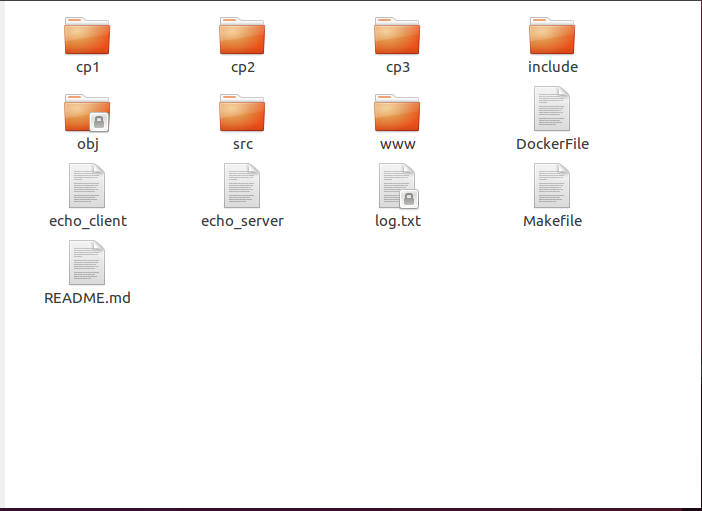


**3.日志**

执行了一些请求后log.txt内容

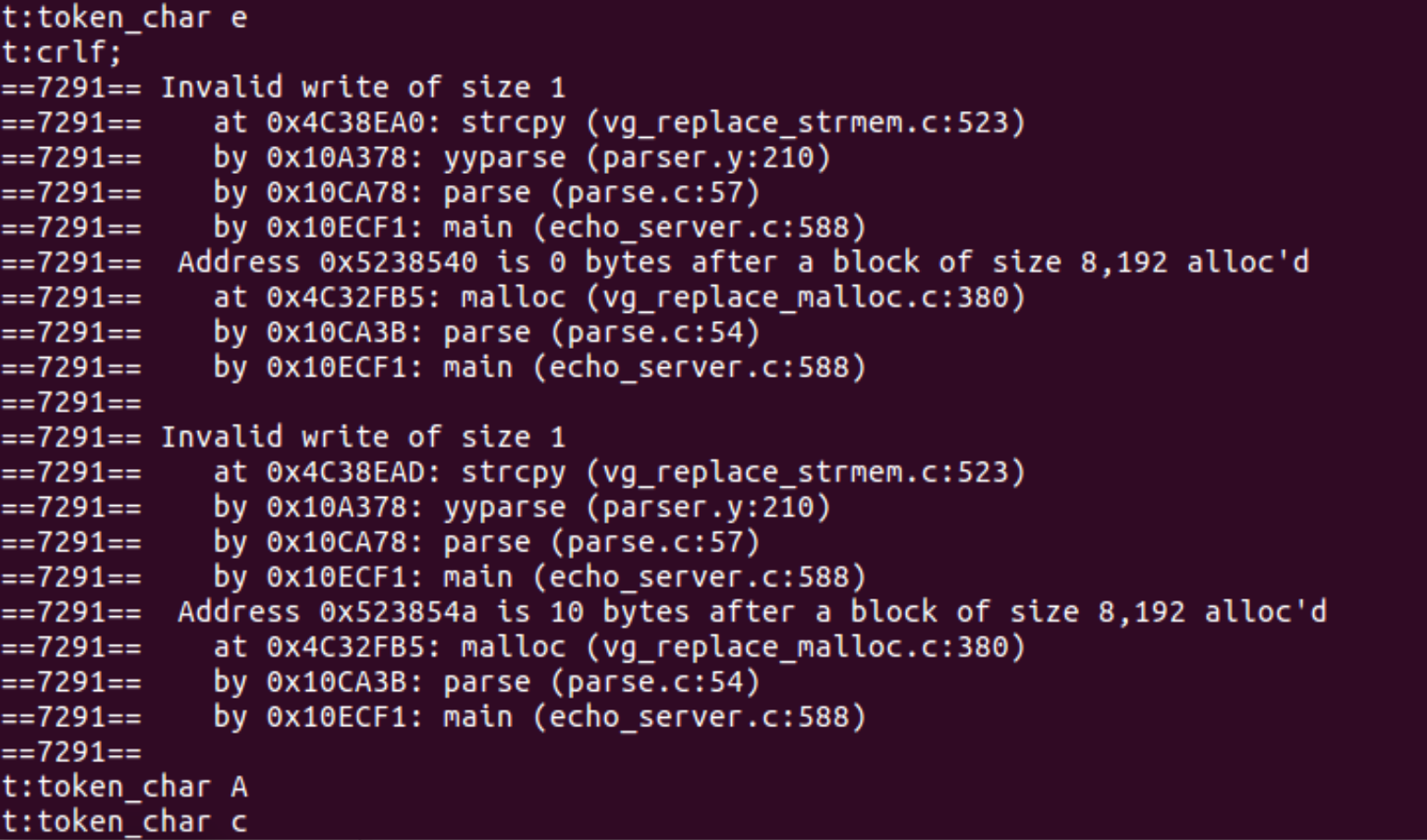


自动生成的日志文件



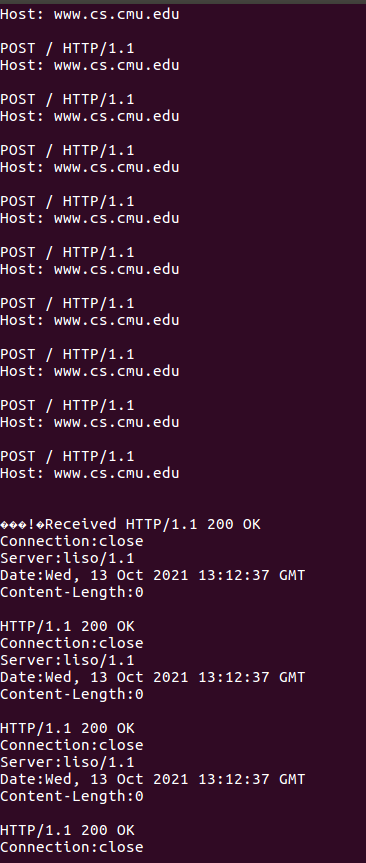
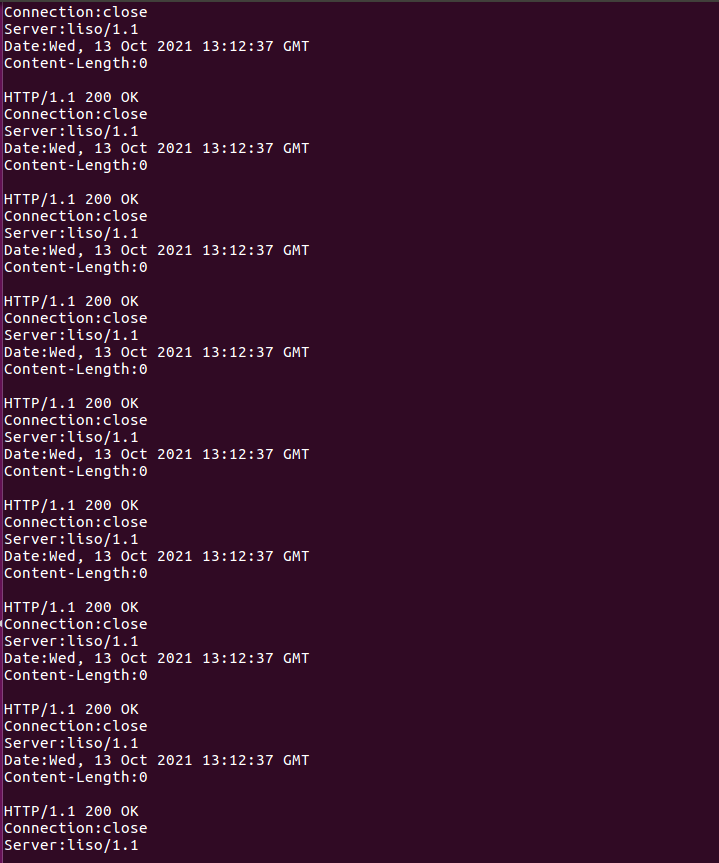
**第三周：**

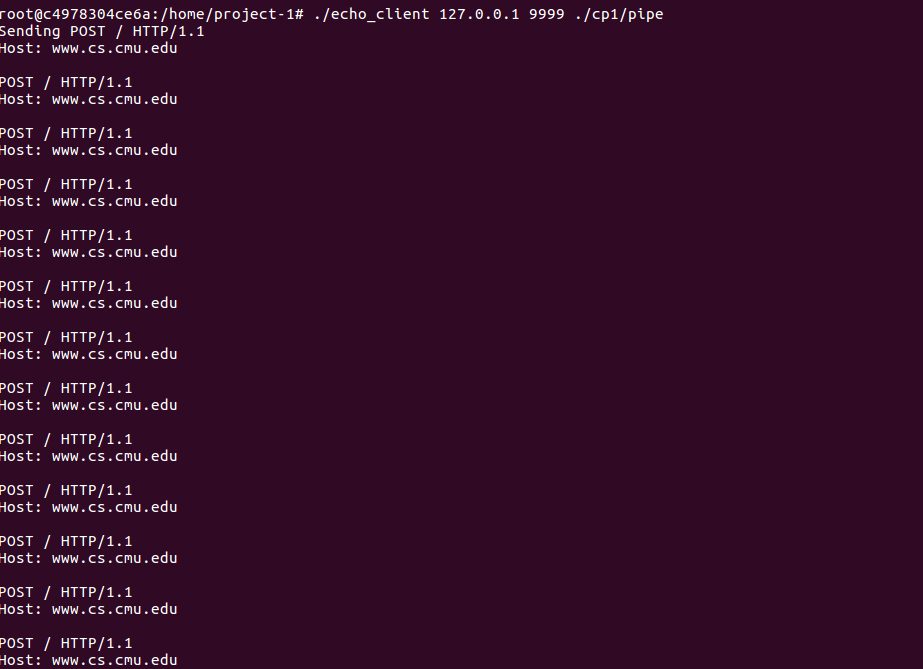
**1.使用Valgrind检查内存泄露过程**

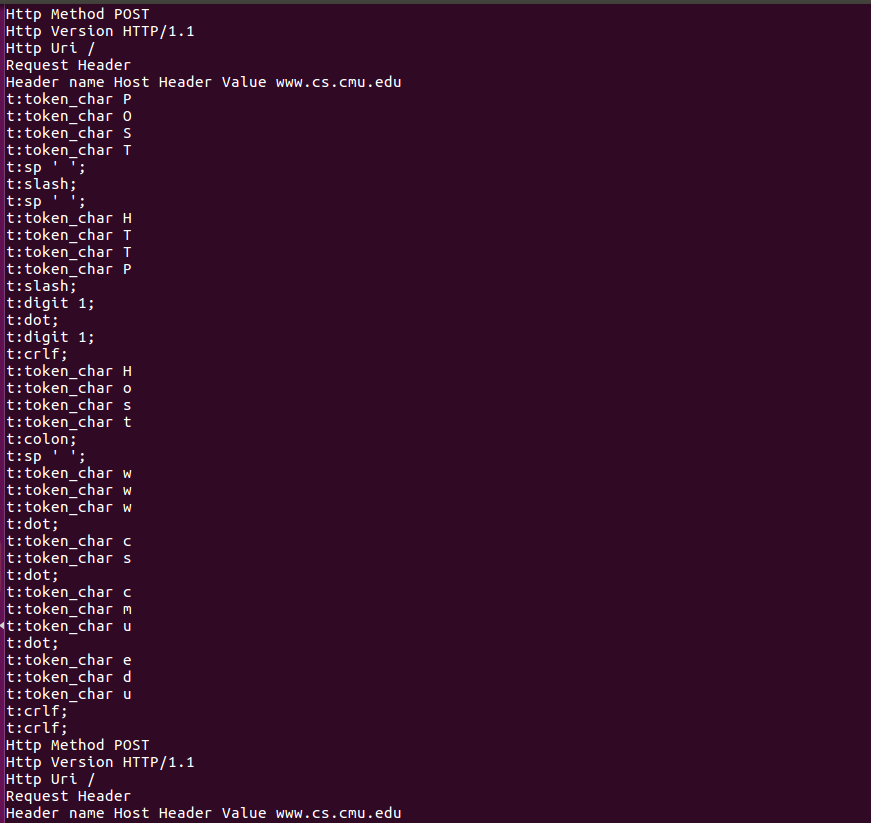


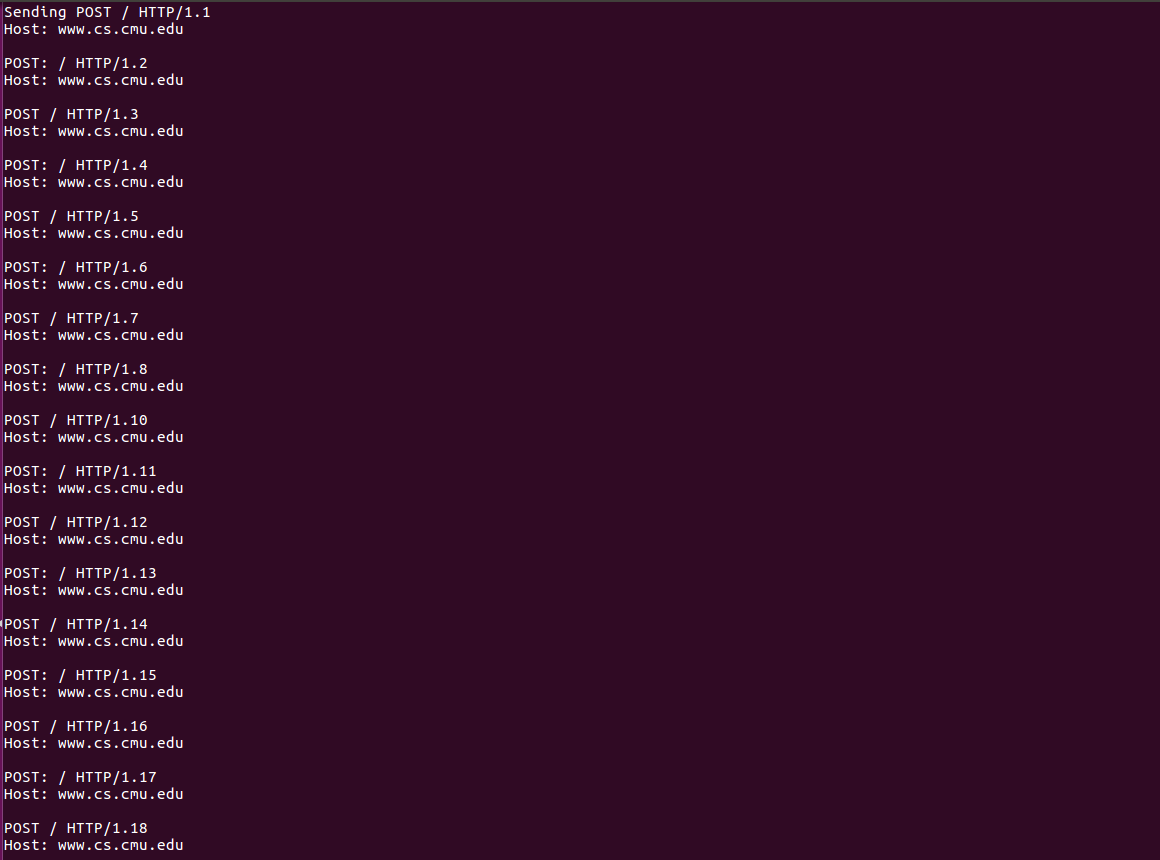
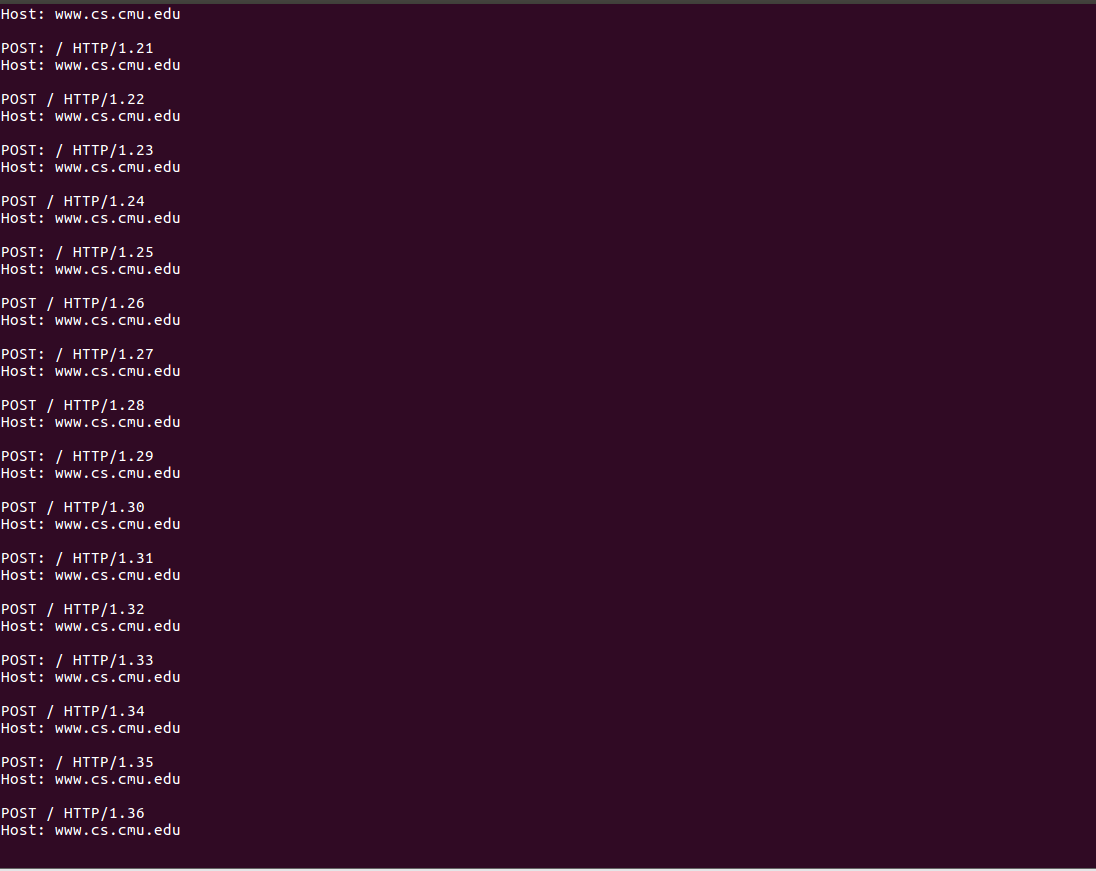
**2.并发请求实现情况（正确解析20个）**

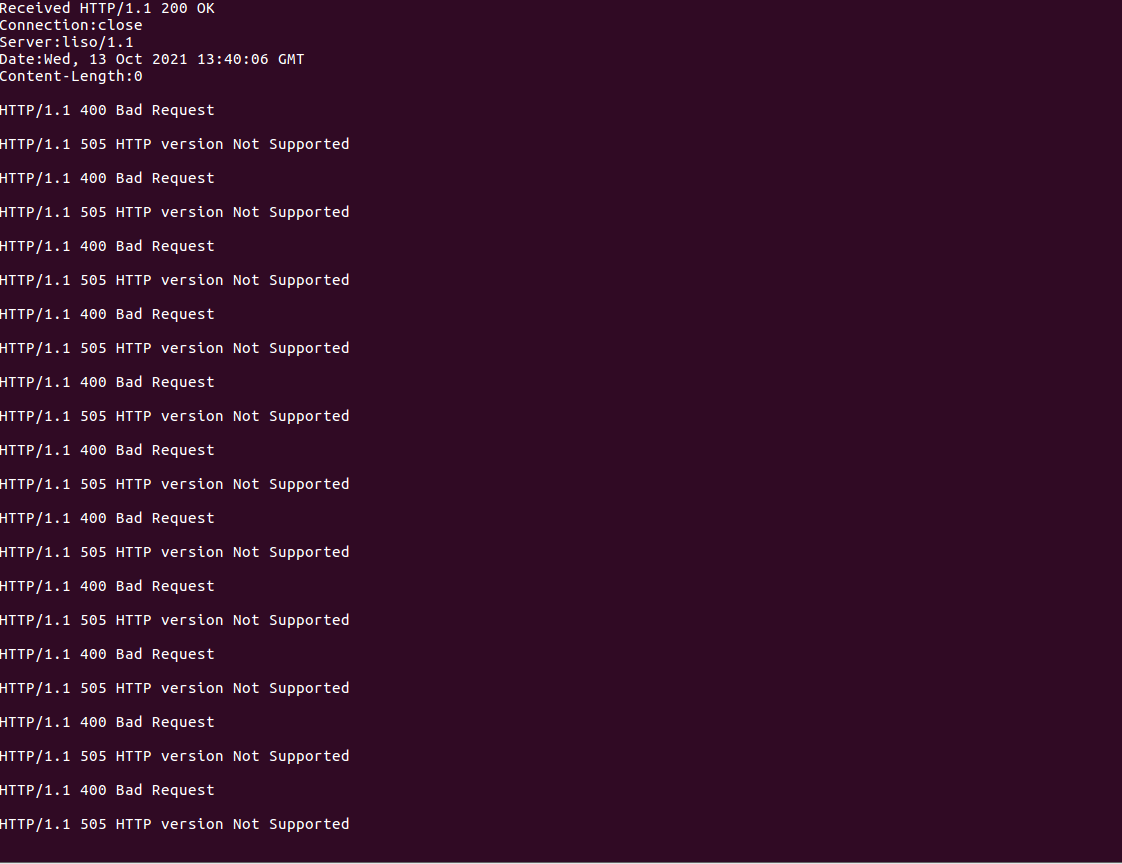
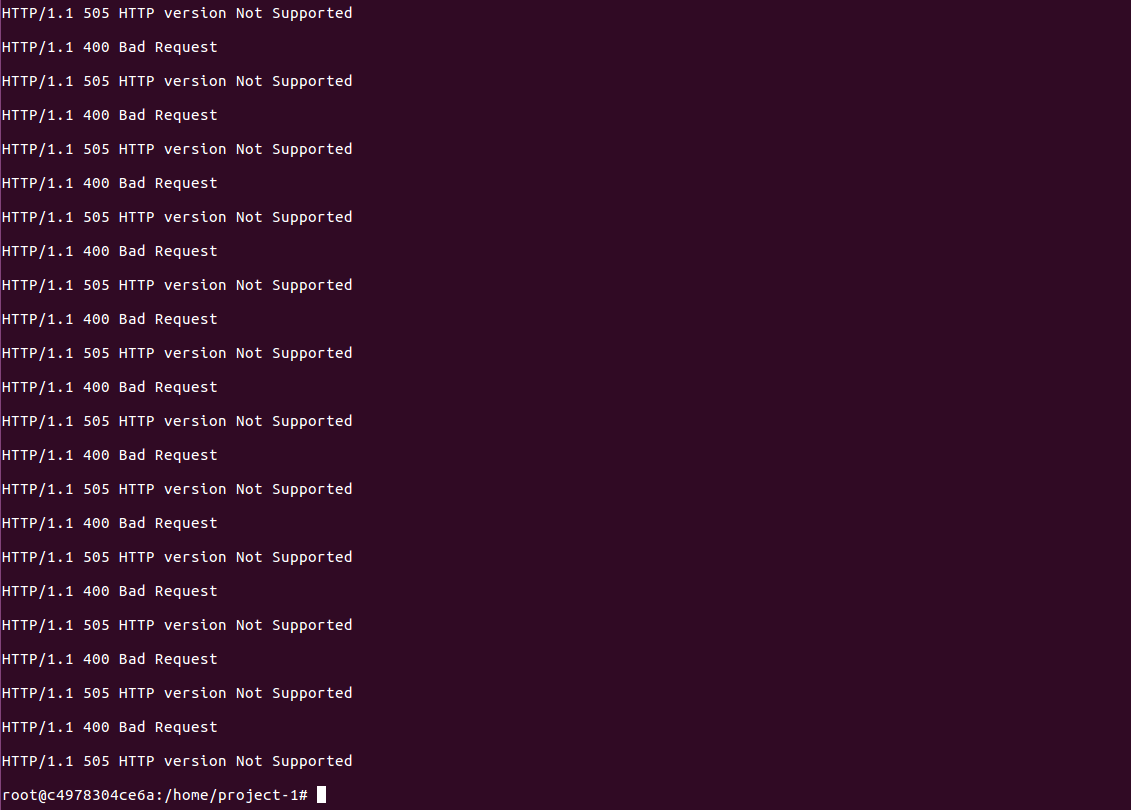
HTTP Pipelining实现结果截图

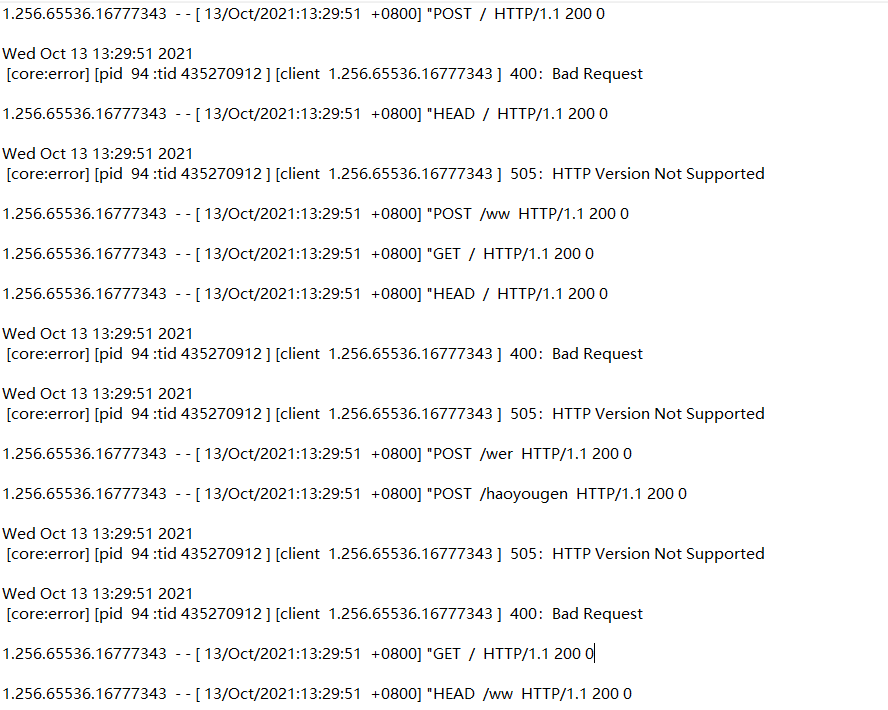


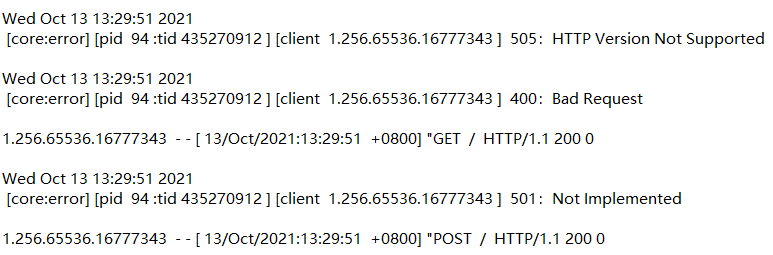


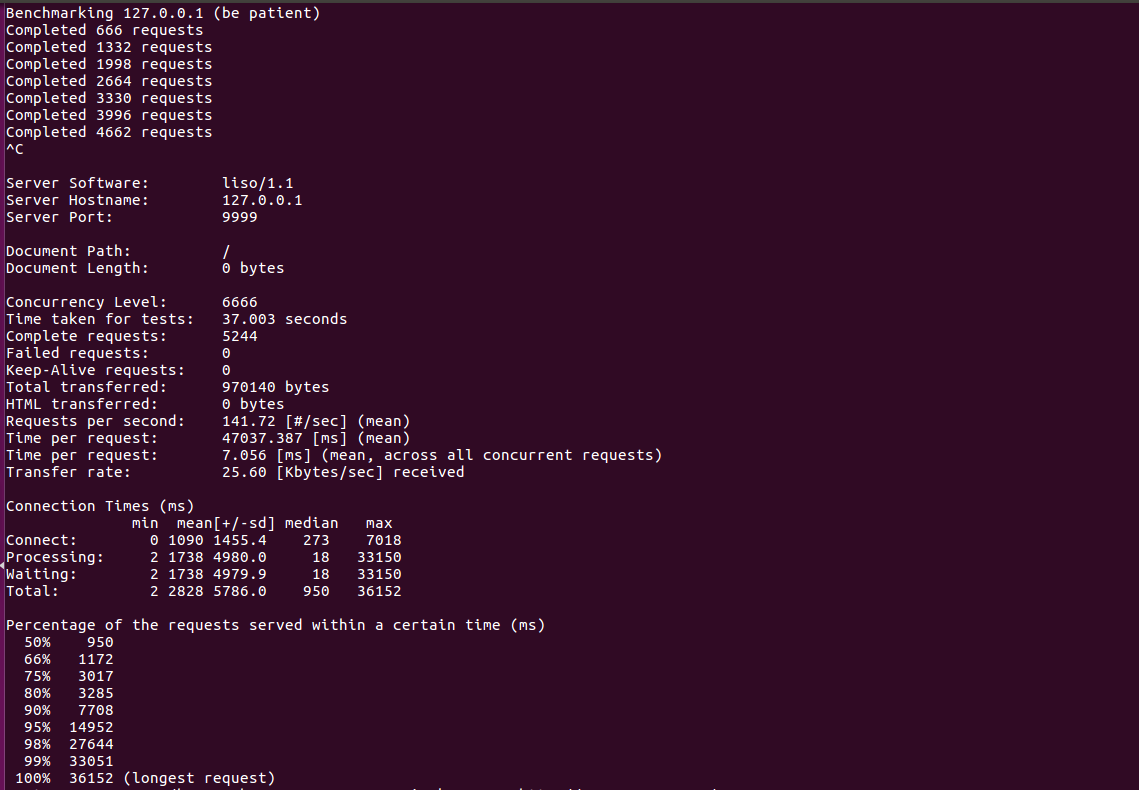
日志体现并发请求的顺序处理（20个）



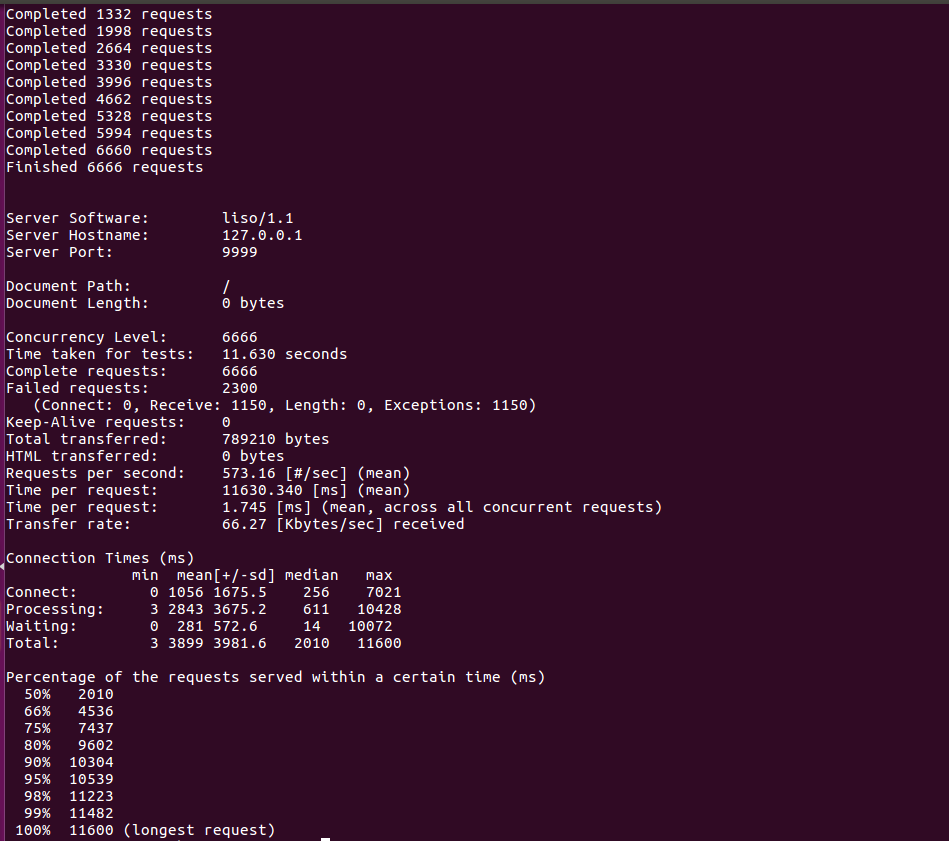


**第四周：**

**1.第一次实验：能够实现5244个并发的客户端**



**2.第二次实验：能够实现5516（6666-1150）个并发的客户端**



**3.CGI用户界面**

**有齐全的环境变量，并完成了一个具有表单的 HTML 页面，能够根据用户在浏览器提交的信息，动态生成网页**

