**《计算机网络》课程设计报告**





**Web Server的设计与实现**

**学 号 3020244294**

**姓 名 石子跃**

**学 院 智能与计算学部**

**专 业 网络空间安全**

**年 级 2020**

**任课教师 仇超**

**2022年 4 月 14 日**

# 一、报告摘要

使用 BSD Socket API（Berkeley Sockets API）实现具有并发性的 web server。依据 RFC 2616 文档[1]，实现 HTTP/1.1 的 HEAD, GET, POST 等基本功能。项目完成后，能够使用普通商用浏览器从所实现的 web server 下载静态网页。

# 二、任务需求分析

1) 能够熟练阅读 Internet 标准 RFC 文件，并实现相应协议；

2) 通过 HTTP 经典协议的复现，掌握应用层协议的设计和实现方法；

3) 掌握应用层协议的并发设计和实现技术。

# 三、协议设计

## **3.1 总体设计**

1. 文件结构分析

（1) 与解析报文有关的文件：parse.h, parse.c, parser.y, lexer.l。

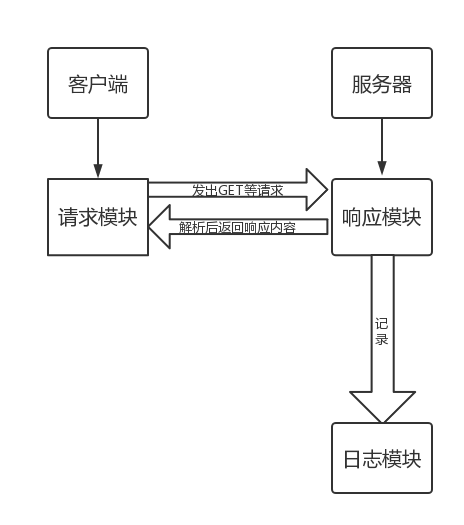
（2）测试环境配置和消息解析的文件：example.c。

（3）模拟客户端和服务端的程序：echo\_client.c（客户端），echo\_server.c（服务端）。

（4）用于测试的数据包样例：samples。

（5）定义编译规则：Makefile。

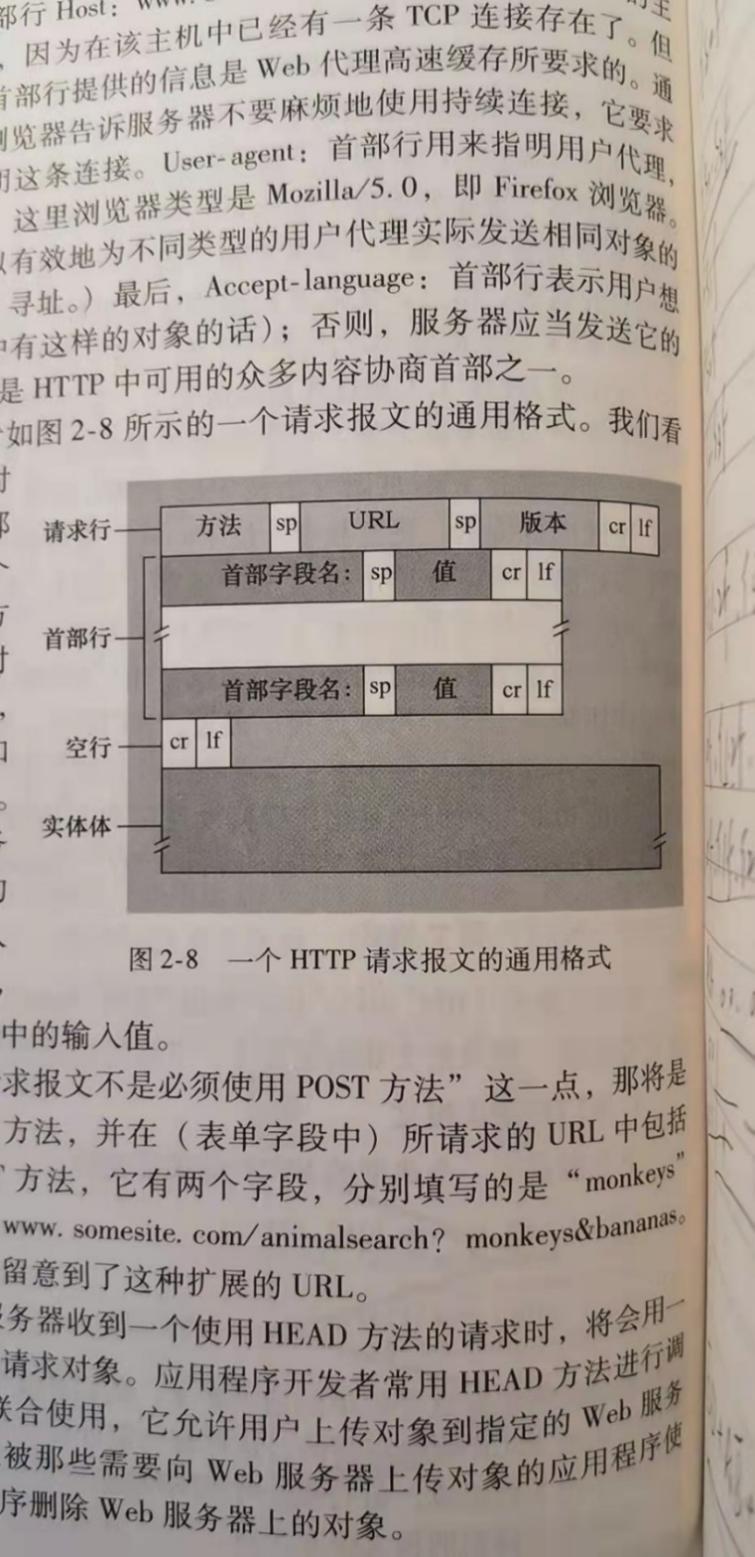
1. 模块组织结构

****

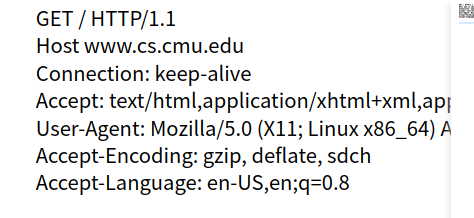
## **3.2 数据结构设计**

1.了解http请求

协议头部、协议规则：



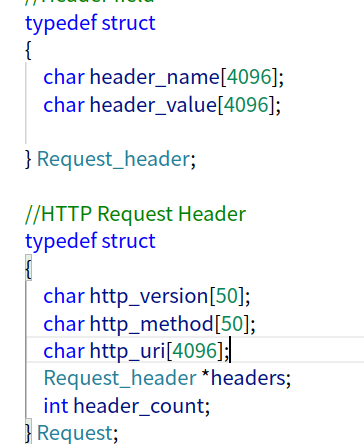
样例：



上图GET / HTTP/1.1 分别对用方法，URL ，版本

2. 数据结构

其中请求行与其他请求头部不一样要单独拿出方便解析。在parse.h中定义http请求的数据结构



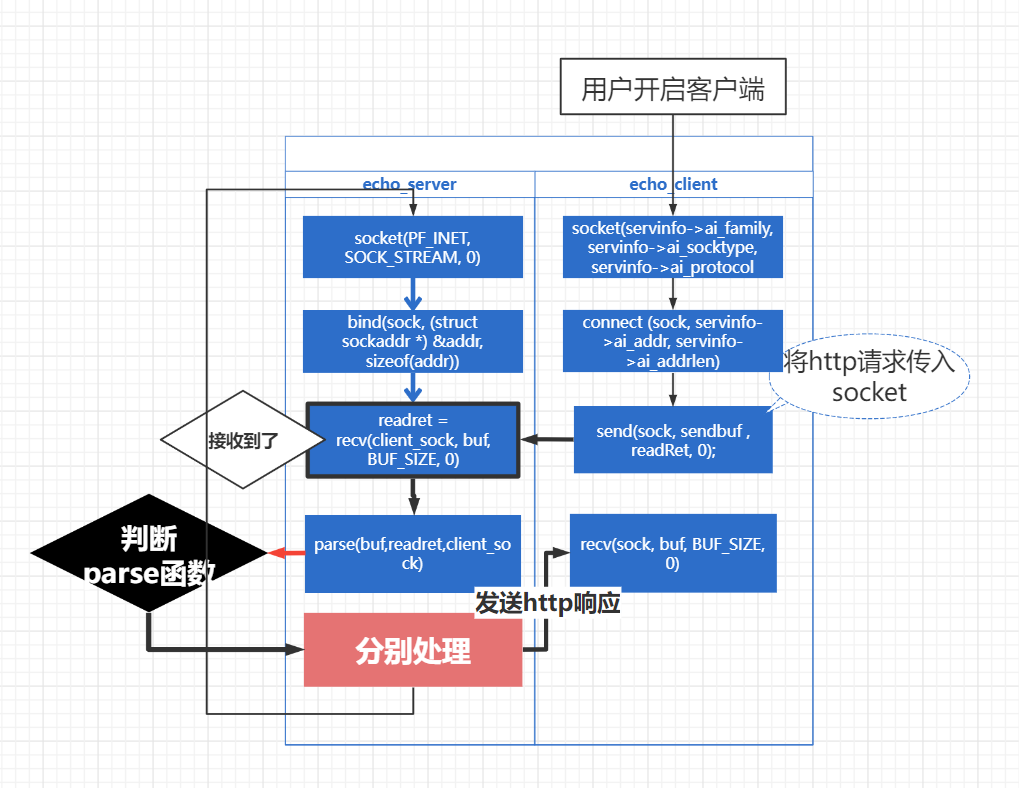
其中http\_version,http\_method,http\_url分别对应分别对应协议规则中的版本，方法，URL，这是http请求行的结构。

## **3.3 协议规则设计**

1. socket原理

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

实现原理如下



服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

这次的代码使用本地ip localhost 127.0.0.1 端口号9999，如果服务器bind失败会提示socket bind failed，如果客户端connect失败会提示connect，在客户端收到一次来自服务器的响应后，echo\_server就会关闭socket，而server还将继续监听这个端口。

# 四、协议实现

## **4.1 简单echo web server的实现**

1). 更改example，parse等文件对http请求案例进行正确解析

首先观察http请求协议的标准格式，学习并根据parser.y，yacc的匹配规则和语法，修改代码使得parse(buf,readRet,fd\_in)函数可以正确解析buf中的字符流。

example.c调用open(),read()函数进行对文件的读，将其字符流保存在buf中。

parse()首先会判断传入的数据中是不是有两个CRLF，这是判断http请求报文最基础的。然后会进入yyparse()，这就是根据parser.y语法文件，所执行的语法解析器。

在request\_header这条规则下，添加它的有递归式使得可以一直连续的识别header。(提示：每个header长得都一样）

同时还要注意对request-headers的动态分配内存以防请求头过多导致溢出。

2). 完善好parse函数后将其移植到echo\_server中，使服务器可正确解析出来自客户端的请求。

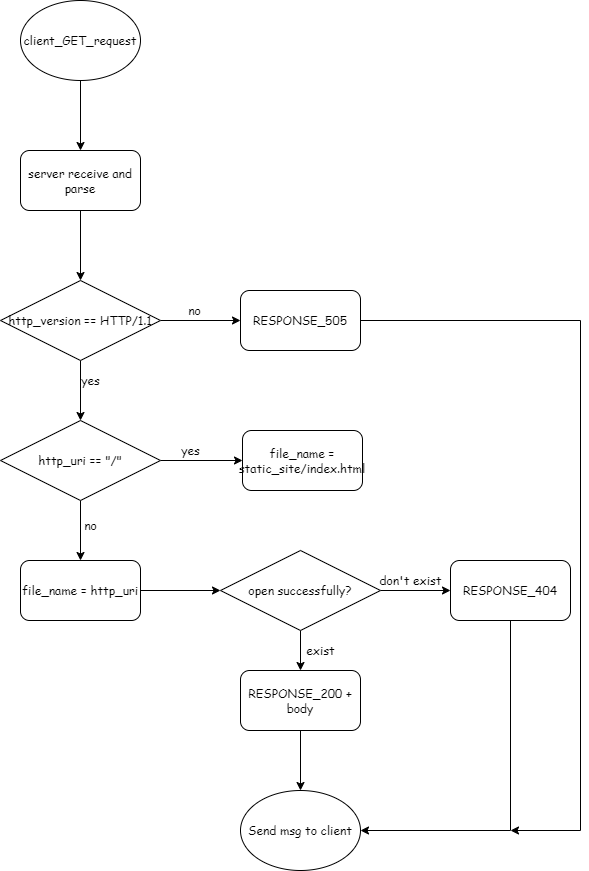
客户端和服务端全部使用buf来缓存数据，当服务器接收到信息后进行解析，再通过send函数返还echo给client，就可开始第三步。echo\_client如何读取文件可以直接从example.c找到。其中由于server调用了parse函数，还需要在makfile中进行相应修改。在进行编译时，将与解析有关的.o文件和server进行连接，生成可执行文件。

3). 服务器对不同请求做出不同响应

阅读http手册，和RFC手册，了解不同状态码对应的含义，在parse中对畸形的请求进行筛选，在echo\_server中对为未部署和已部署的request\_method进行不同的响应，比如501，400等。我目前的做法是畸形的请求交给parse.c处理，未部署的请求放在eecho\_server中处理。

## **4.2 HEAD、GET、POST方法的实现**

1) GET请求的响应过程



2). Head方法：与上述Get方法不同，只需根据url中的文件地址去搜索文件，此方法被用来获取请求实体的元信息而不需要传输实体主体（entity-body）。此方法经常被用来测试超文本链接的有效性，可访问性，和最近的改变。更改报文中的响应状态行，Content-Length，Content-Type，Date，Last-modified等

3). Post方法：被用于请求源服务器接受请求中的实体作为请求资源的一个新的从属物。本次实验只需要echo回去post请求本体就可，但是我实现了echo回需要post的数据。

4). 支持 5 种 HTTP 1.1 出错代码：400，404，408，501，505。准确判别客户端消息，并发回响应消息。400 = "HTTP/1.1 400 Bad request " 404 = "HTTP/1.1 404 Not Found " 501 = "HTTP/1.1 501 Not Implemented " 505 = "HTTP/1.1 505 HTTP Version not supported”，查看httprfc手册。

5). 持久连接（persistent connection）：长连接的退出动作由客户端发出，也就是利用connection头域可以产生终止连接的信号，在客户端发出带有close的一条指令后，服务器也会返回响应状态connection：close，在完成响应报文send后，客户端便不可再想此连接提出任何新请求。

6). 创建简化的日志记录模块，记录格式化日志。

apache的access\_log日志格式样例：

i. 127.0.0.1 - - [21/Mar/2022:20:55:14 -0700] "GET / HTTP/1.1" 304 -

ii. 127.0.0.1 - - [21/Mar/2022:20:55:48 -0700] "GET / HTTP/1.1" 200 45

iii. 127.0.0.1 - - [21/Mar/2022:20:55:48 -0700] "GET /favicon.ico HTTP/1.1" 404 196

iv. 127.0.0.1 - - [23/Mar/2022:06:21:15 -0700] "GET / HTTP/1.1" 200 45

v. 127.0.0.1 - - [23/Mar/2022:06:21:15 -0700] "GET /favicon.ico HTTP/1.1" 404 196

于是自己定义新的格式如下：

（时间，host，method，url，响应代码）

7).具体实现方法

1. GET方法，HEAD方法，POST方法集成在response.c中，将server需要使用的命令在response.h中。相比于我最初把这三个方法分别编写.c.h文件，只用一个.c文件更好，避免了文件之间函数的重复引用，更简单明了。

2. GET：客户端会传递url来访为相应位置的文件，于是编写以下函数：

a) get\_file\_size(char\* file\_name) 获取响应文件大小

b) get\_file\_type(char\* filename) 获取文件类型

3. HEAD：

这是一个head请求的响应格式，观察得到Content-type，Content-length在get中有所实现，但Date，Last-modified首部行还未实现，于是编写函数如下：

a) getime() 获取当前时间

b) getlast(char \*path) 获取文件最后修改时间

4. Post：起初实现的可以根据Content-Length，返回已post的函数，考虑到首部行中传输的格式为ASCII，于是编写了一个int char2int(const char\* str)函数，来读出文本长度，最后，echo回客户端

5. 持久连接：大致实现如下：

需要编写函数：

addalive(Request \*request,char\* buff) 用于当服务器接收到带有connection：keep-alive的请求头部时进行正确的操作

isalive(char\* buff) 用于客户端正确识别服务器是否支持继续持久连接

## **4.3 HTTP的并发请求的实现**

1) 分解请求

在从client传入的数据中，按照每个HTTP请求的格式发现最后都以\r\n结尾。于是可以新建一个char指针去寻找其位置，并将中间的数据传入parse中进行解析，当服务器对当前请求进行响应后，将指针继续向后寻找，重复上述动作。在指针发现位置已经到达其尾部则退出当前循环，让server准备好recv接下来的请求。

2) 具体实现

定义char \*index用来指向\r\n\r\n。

使用Strstr函数来定位准确位置。

发现psrse函数还需传入一个int size于是需要一个long int 变量length来记录单独的一个请求报文长度。

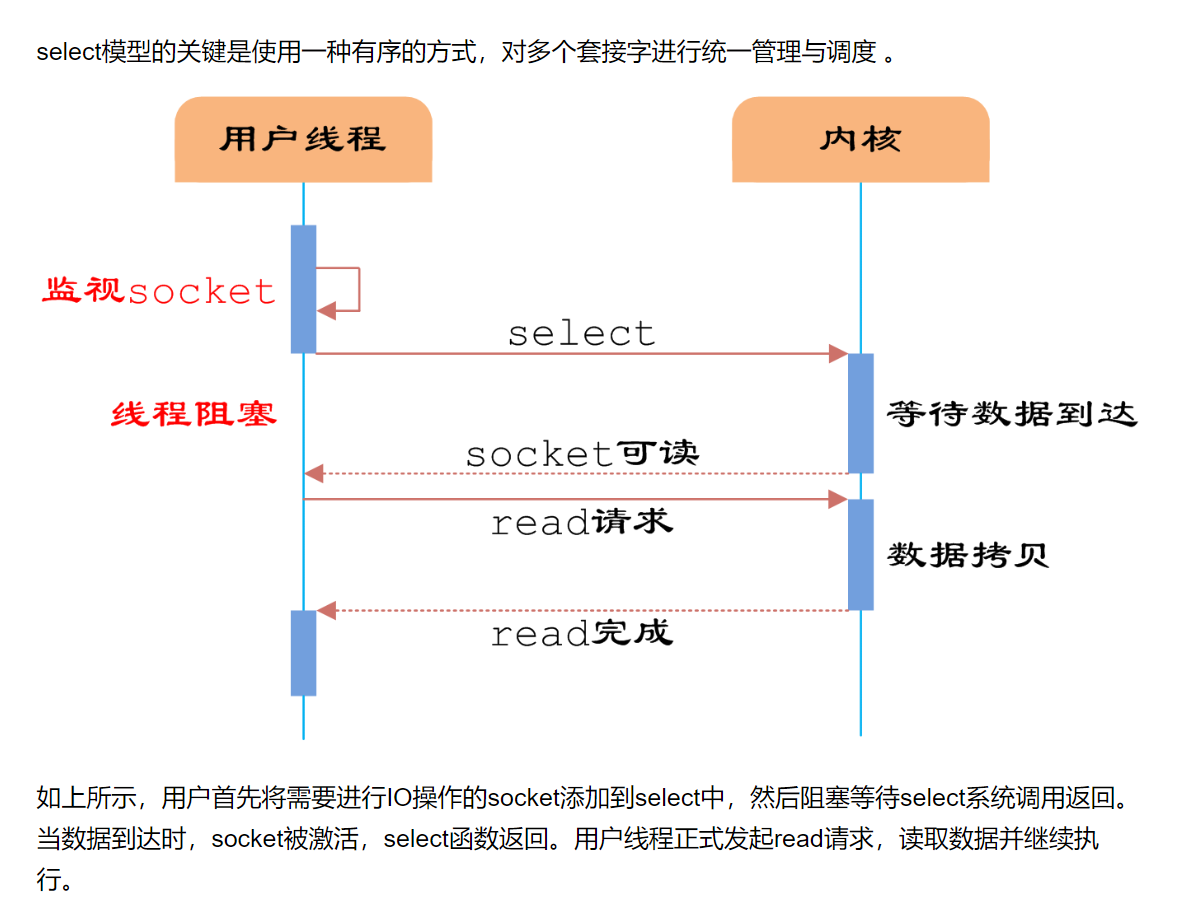
注意每次提取request把\r\n\r\n这四个字节加上

While（当前读到的位置还未到此次接受的并发请求的长度）就继续协议设计中的步骤。

注意使用send函数时，发送的长度最好使用strlen(buf)如果直接传递BUF\_SIZE可能会因为后续客户端recv不够及时导致缓冲区内数据过多，后续的send函数不能进入。

## **4.4 多个客户端的并发处理**

1） 了解select方法



int select(int maxfdp, fd\_set \*readset, fd\_set \*writeset, fd\_set \*exceptset,struct timeval \*timeout);

int FD\_ZERO(int fd, fd\_set \*fdset); //一个 fd\_set类型变量的所有位都设为 0

int FD\_CLR(int fd, fd\_set \*fdset); //清除某个位时可以使用

int FD\_SET(int fd, fd\_set \*fd\_set); //设置变量的某个位置位

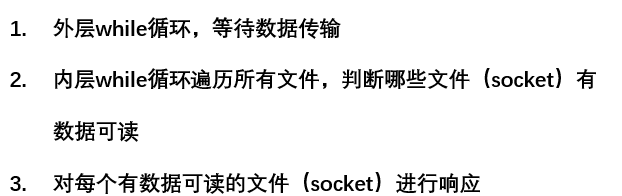
int FD\_ISSET(int fd, fd\_set \*fdset); //测试某个位是否被置位

select模型的关键是使用一种有序的方式，对多个套接字进行统一管理与调度 。当声明了一个文件描述符集后，必须用FD\_ZERO将所有位置零。

用户首先将需要进行IO操作的socket添加到select中，然后阻塞等待select系统调用返回。当数据到达时，socket被激活，select函数返回。用户线程正式发起read请求，读取数据并继续执行。

从流程上来看，使用select函数进行IO请求和同步阻塞模型没有太大的区别，甚至还多了添加监视socket，以及调用select函数的额外操作，效率更差。但是，使用select以后最大的优势是用户可以在一个线程内同时处理多个socket的IO请求。用户可以注册多个socket，然后不断地调用select读取被激活的socket，即可达到在同一个线程内同时处理多个IO请求的目的。而在同步阻塞模型中，必须通过多线程的方式才能达到这个目的。

2） 具体实现



FD\_SET(sock, &readfds); //将服务器端socket加入到集合中

for (fd = 0; fd < FD\_SETSIZE; fd++)/\*扫描所有的文件描述符\*/

if (FD\_ISSET(fd, &testfds)) /\*找到相关文件描述符\*/

if (fd == sock)/\*找到服务器程序的文件描述符\*/

else/\*是客户端socket则对其进行响应

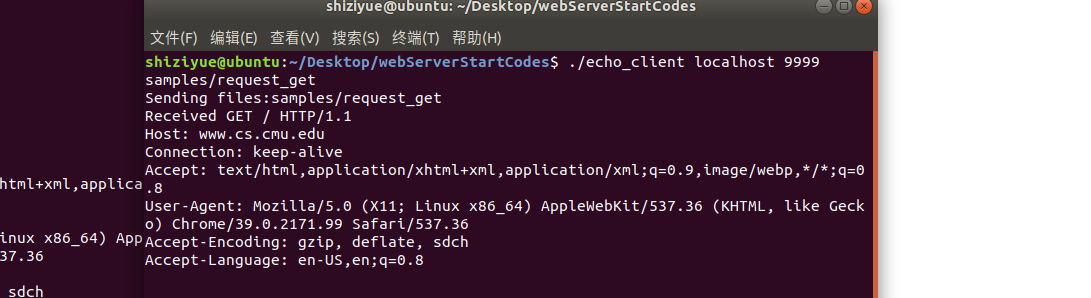
# 五、实验结果及分析

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。

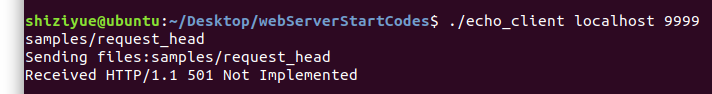
## **5.1 简单echo web server的实验结果与分析**

实验结果：

1. 如果收到客户端发来的是 GET, HEAD 和 POST 方法，则 echo 回去，即重新封装（encapsulation）消息并返回给客户端。

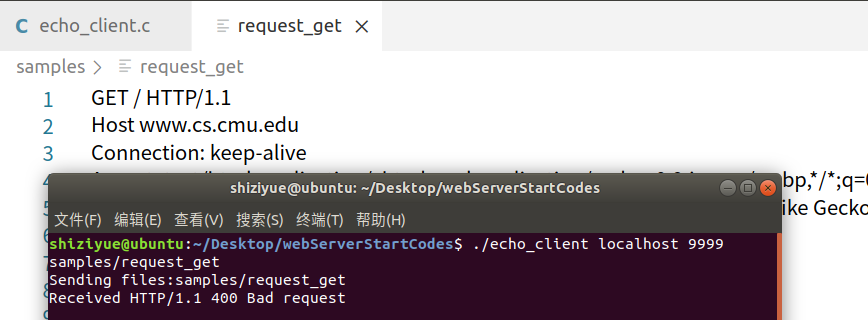


2） 如果收到客户端发来的是除 GET, HEAD 和 POST 以外的其它方法，服务 器 并 没 有 实 现 ， 则 需 要 返 回 响 应 消 息 “ HTTP/1.1 501 Not mplemented\r\n\r\n”。



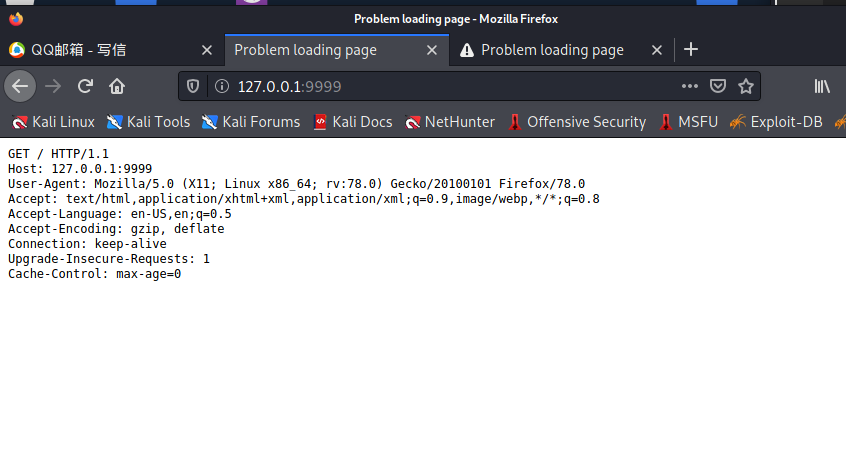
3） 如果收到的客户端消息的格式错误，应能够识别出来，并返回错误代码

为400的HTTP 响应消息“HTTP/1.1 400 Bad request\r\n\r\n”



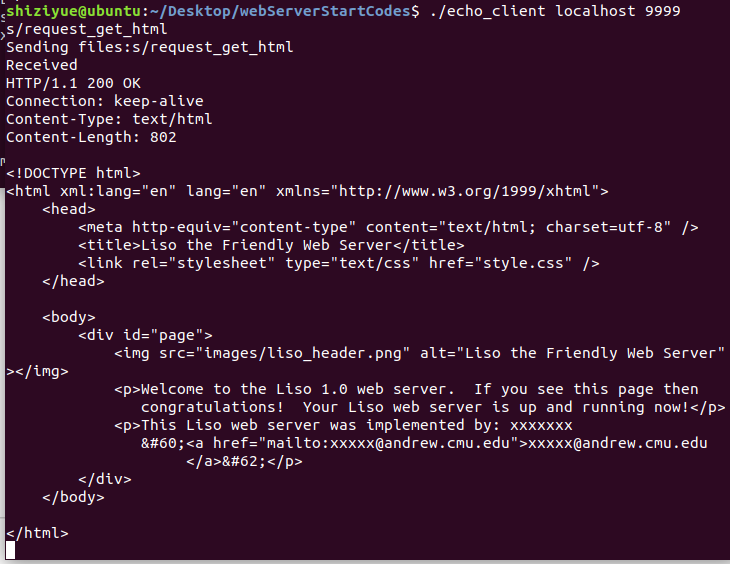
1. 浏览器测试

在服务器开启的时候，浏览器访问127.0.0.1:9999，得到以下的get报文， 返回了get请求报文



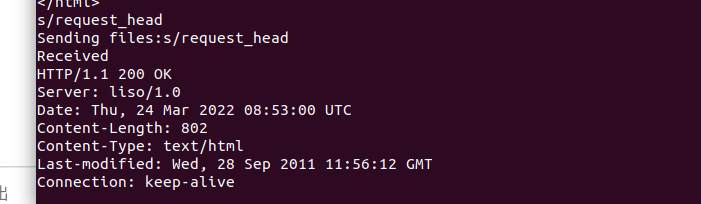
## **5.2 HEAD、GET、POST方法的实验结果与分析**

1） Get



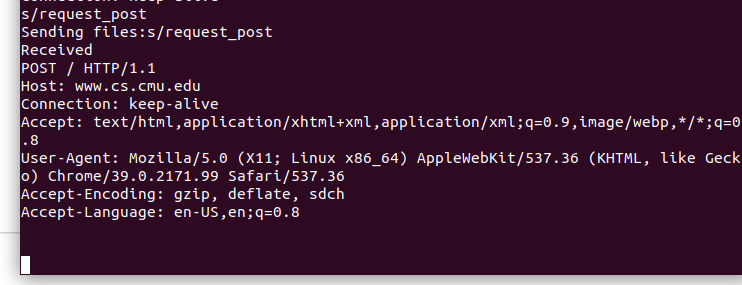
结果和上文描述一致，同时客户端可继续向server发送请求。也实现了持久连接。

2） Head：

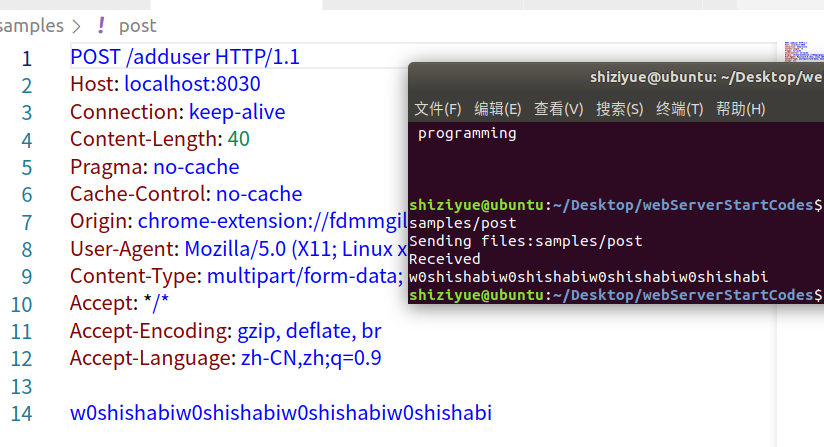


返回和上文一中叙述一致。

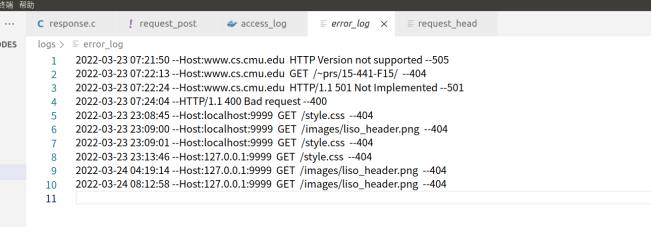
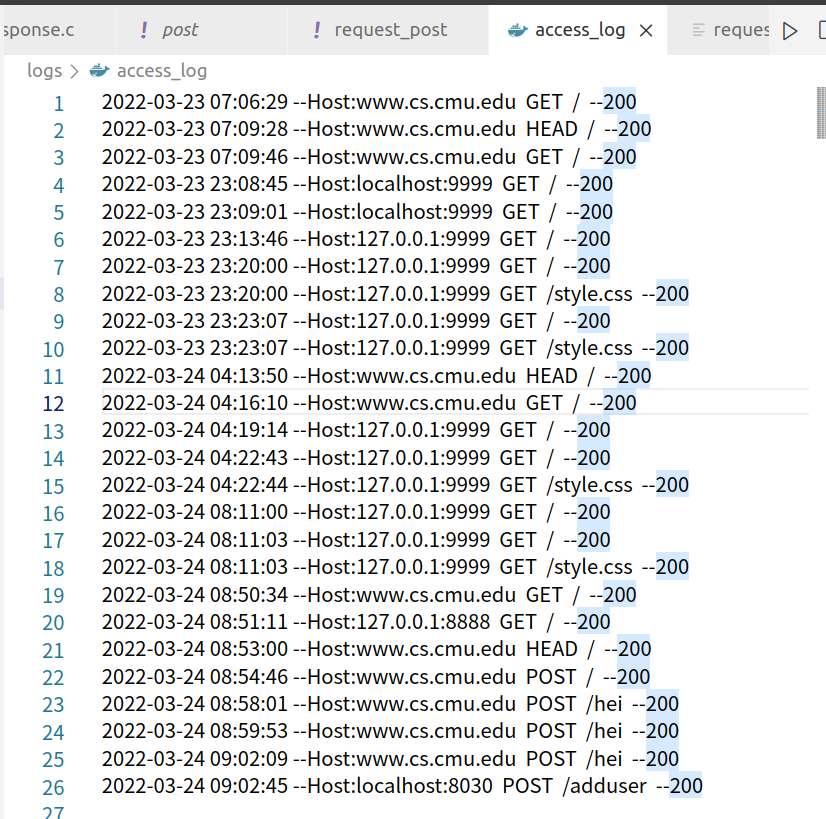
3） Post：



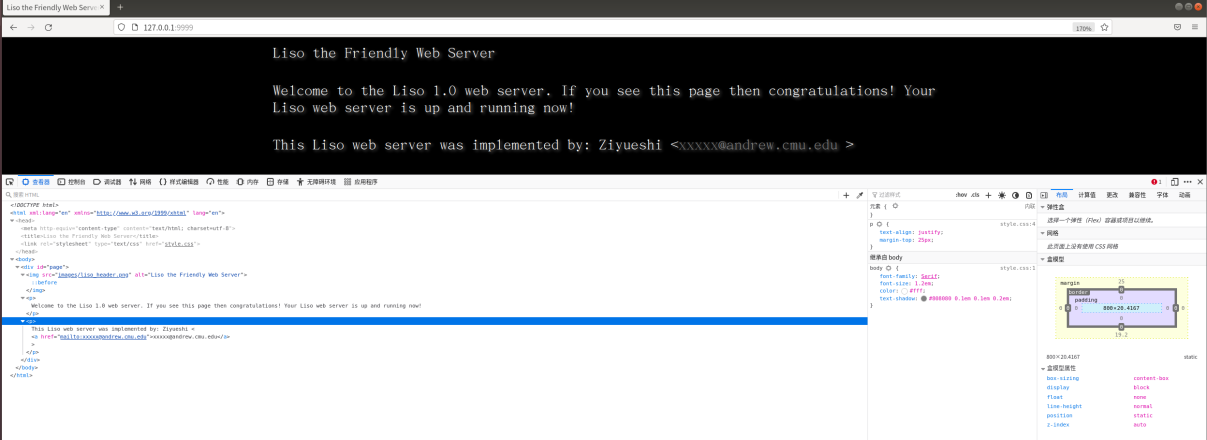
相比于本次实验要求的我还新增了一个功能，即可将post的数据返回



4）Log日志



5） 最后浏览器运行测试



## **5.3 HTTP的并发请求的实验结果与分析**

1） 以所给的并发请求样本文件samples/request\_pipeline为例做本地测试，以下为回显，成功实现

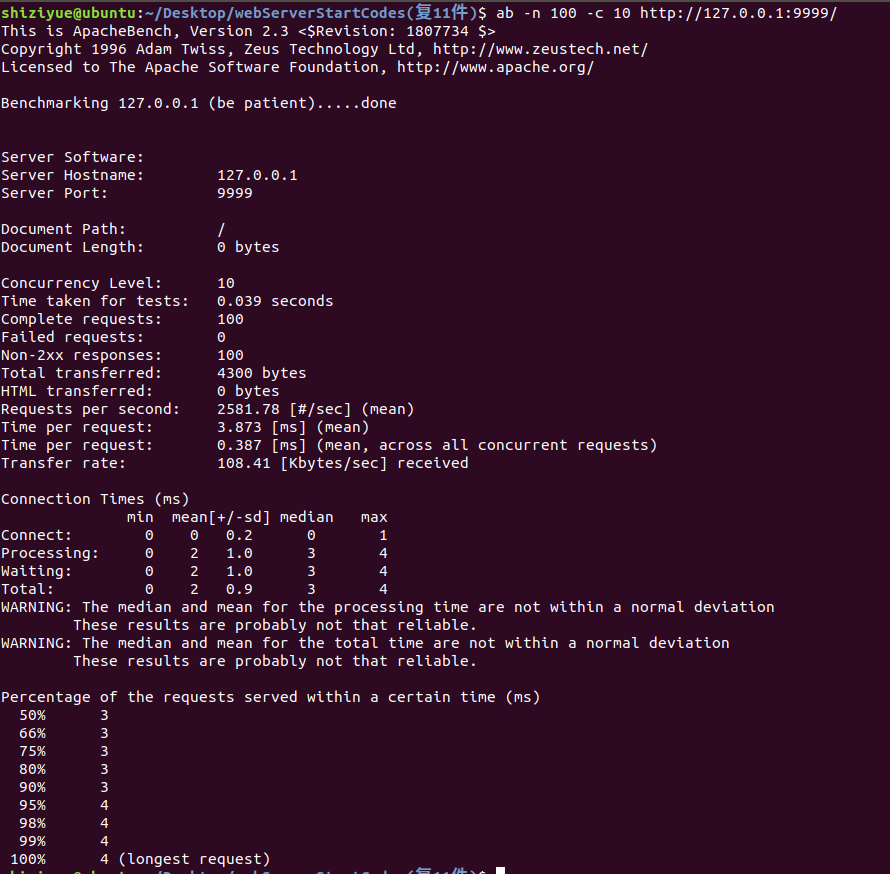


## **5.4 多个客户端的并发处理的实验结果与分析**

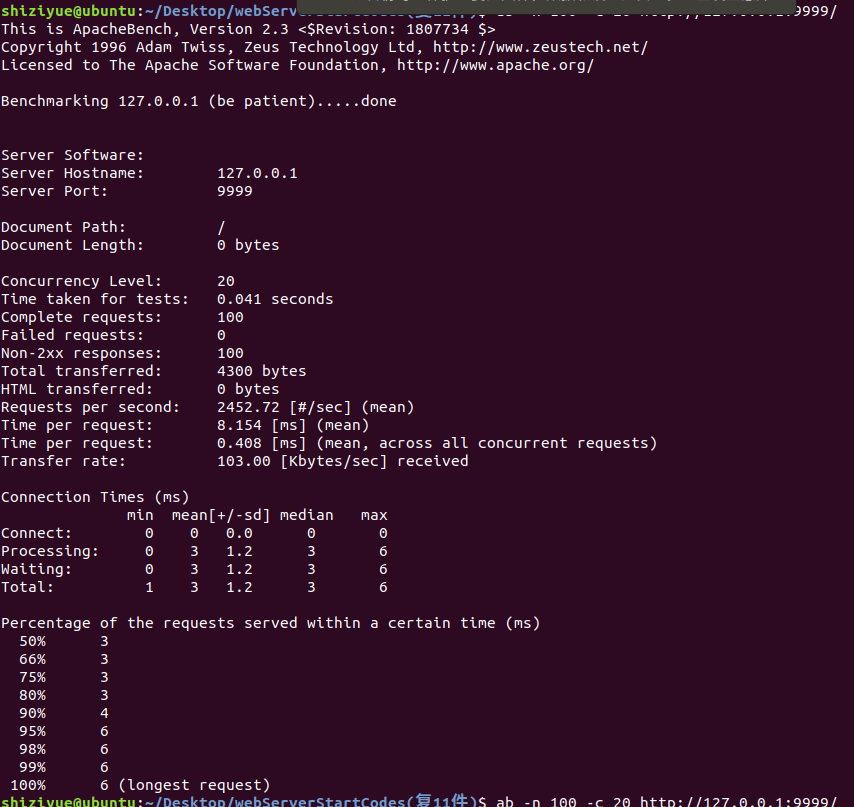
1） 实验结果

实现了select的多线程之后，发现相比于实验3中的代码速度改进许多，

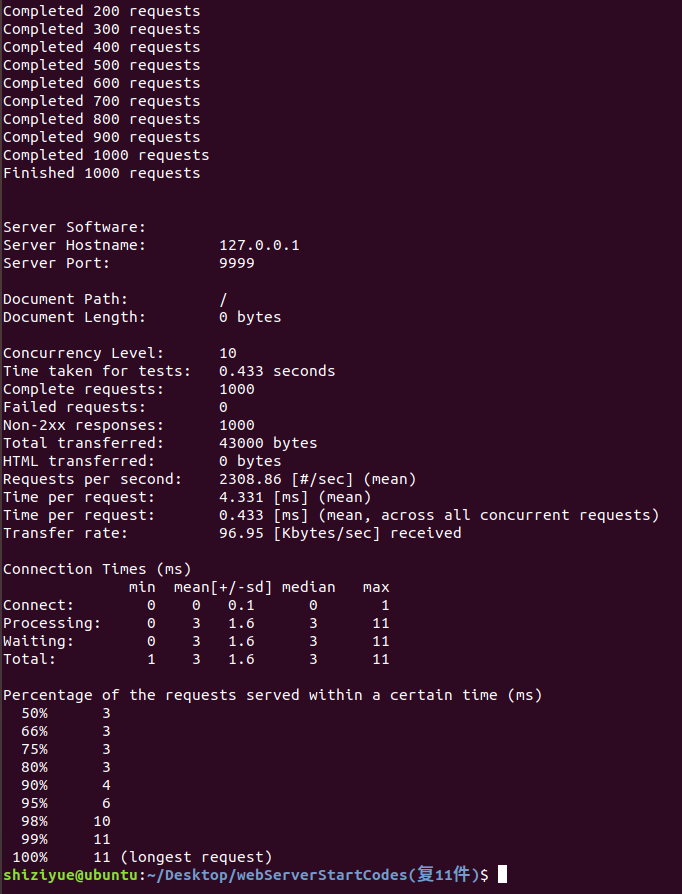
下面是共计发送总计100个，每次十个并发的结果



下面是共计发送总计100个，每次20个并发的结果



下面是共计发送总计1000个，每次10个并发的结果



2）结果分析

发现总数一样时，改变并发数，响应时间随着并发数增加而增加，当并发数一样时，响应时间随总数增加而增加。

作用：

使用select以后最大的优势是用户可以在一个线程内同时处理多个socket的IO请求。在网络编程中，当涉及到多客户访问服务器的情况，我们首先想到的办法就是fork出多个进程来处理每个客户连接。现在，我们同样可以使用select来处理多客户问题，而不用fork。

但是，对socket进行扫描时是线性扫描，即采用轮询的方法，效率较低：当套接字比较多的时候，每次select()都要通过遍历FD\_SETSIZE个Socket来完成调度,不管哪个Socket是活跃的,都遍历一遍。这会浪费很多CPU时间。还需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大。

# 总结

这次实验的难点主要在于第一周对源代码各个部分的阅读和理解，在最初拿到项目代码的时候面对大量的代码感觉难以理解。面对这种情况，我的解决方法就是结合老师和助教上课讲的内容，抓住重点代码进行深入解读。遇到陌生的函数通过上网进行查询和学习，通过自己的努力以及与队友的相互帮助下，最终快速的理解了代码目的，并且成功实现要求。

除此之外在具体功能的实现上常常会遇见各种奇怪的bug，这时只能耐心的进行调试。这些bug多是由于某一处小小的问题，因此需要从头到尾的检查代码，最终才能完成任务。

通过这次实验，我深入的理解了课本上各种协议的具体实现方法，通过亲自用代码实现协议要求的途径，完全的掌握了书本上的知识点，收益匪浅。