**《计算机网络》课程设计报告**





**HTTP 的设计与实现**

**学 号 3019209298**

**3019207257**

**姓 名 彭恺宁 李自安**

**学 院 求是学部**

**专 业**

**年 级 19级**

**任课教师 赵增华**

**年 月 日**

# 报告摘要

HTTP协议需要解决WEB端内容的获取和传输问题。协议设计过程中需考虑数据传输的安全性及可靠性，同时还需考虑在大量信息传输时服务器的稳定性。本周我们实现了简单的echo web server，服务器端在收到客户端发送的消息后能够正确解析并返回响应信息，根据客户端发送消息的不同，服务器端返回的响应消息分为3类，分别为：原消息、错误代码为501的HTTP响应消息和错误代码为400的HTTP响应消息。

报告主要包括协议的需求分析、协议设计、协议实现、实验结果及分析与个人总结。在需求分析部分，我们从数据传输、网络服务、设计目标等层面进行了HTTP协议的需求分析；在协议设计部分，我们确定了HTTP协议的架构、传输层协议、基本规则与数据结构设计、首部设计、方法设计与相应码设计。

# 二、任务需求分析

## 1. 数据传输需求

HTTP主要应用于WEB端内容的获取和传输，其定义了Web客户向Web服务器请求Web页面的方式，以及服务器向客户传送Web页面的方式。这意味着HTTP需要一种**可靠的数据传送服务**，同时，数据的传输也需要**安全性**的保证。而HTTP在进行Web数据传输时，对**时间是不敏感的**。此外，在进行信息传输时，应当考虑到HTTP对**吞吐量、并发用户数目**的需求，即应当在大量信息传输、多用户同时请求时保持服务器的稳定性。

## 网络服务需求

在进行数据传输时，HTTP主要由客户端向服务端发起请求，建立连接。因此，HTTP需要一种**能够提供稳定连接的网络服务。**

## 设计目标：性能指标

HTTP协议的设计应当使得以下性能指标达到尽量的优化，具体指标包括以下几点：

1. **请求时间**：用户从三次握手到最后一次请求发出的这一段时间。

2. **网络丢包率（loss）**：当前的网络丢包情况统计。

3. **网络时延（delay）**：当前的网络时延，主要为RTT的大小。

4. **吞吐率（throughput）**：这里的吞吐率特指Web服务器单位时间内处理的请求数，也可以用来描述其并发处理能力。

5. **并发用户数**：严格意义上的并发为一定数目的用户在同一时刻做同一件事情或操作，广义上的并发可以指在线用户的数目（尽管实际上在线用户不一定会和其他用户发生并发）。

除却以上指标以外，还有以下几个与响应时间（响应速度）相关的指标：

1. 页面加载时间
2. 全部页面加载时间
3. 首字节时间
4. 使用长连接
5. DNS时间
6. TCP时间

在设计过程中，应当本着使得与相应时间相关的指标、丢包率、网络时延越小越好；吞吐量、并发用户数等指标越高越好。

# 协议设计

## 1. 协议架构

由于HTTP协议主要定义了Web客户向Web服务器请求Web页面的方式，以及服务器向客户传送Web页面的方式，数据与内容存储于服务器端，因此应当采用**Client-Server(C/S)**结构，由服务器负责数据的管理，客户机负责完成与用户的交互任务。

## 2. 传输层协议

由HTTP的需求分析可知，我们应当选择**TCP传输层协议**，这是因为TCP能够保证以下几点要求：

1. 传输可靠
2. 丢包率低
3. 传输数据量大
4. 面向连接

而相比之下，HTTP对传输速度的要求并不是很高，因此TCP协议是十分合适的。

## 3. 基本规则与数据结构设计

基本的解析结构如下设计：

OCTET（字节） = <任意八比特的数据序列>

CHAR = <任意ASCII字符（ascii码值从0到127的字节）>

UPALPHA = <任意大写字母>

LOALPHA = <任意小写字母>

ALPHA = UPALPHA | LOWALPHA

DIGIT = <任意数字0，1，…，9>

CTL = <任意控制字符（ascii码）>

LF = <US-ASCII CR, 回车>

SP = <US-ASCII LF，换行符>

HT = <US-ASCII HT，水平制表符>

<"> = <US-ASCII，双引号>

**具体规则**应当包括以下几种：

**Rule 1: Allowed characters in a token.**

token = 1\*<any CHAR except CTLs or separators>

**Rule 2: A token is a sequence of all allowed token chars.**

**Rule 3: Allowed characters in text.**

TEXT = <any OCTET except CTLs, but including LWS>

**Rule 4: Text is a sequence of characters allowed in text as per RFC. May also contains spaces.**

**Rule 5: Optional white spaces.**

具体的请求如下规定：

**request\_line: token t\_sp text t\_sp text t\_crlf**

**request\_header: token ows t\_colon ows text ows t\_crlf｜**

**request\_header token ows t\_colon ows text ows t\_crlf**

**request: request\_line request\_header t\_crlf**

## 4. Header（首部）设计

根据HTTP数据交互的要求，应当在HTTP Header中进行Request、Response、Entity以及其他通用的设置的信息规定。主要类型包括有：常用头域、请求头域、响应头域、实体头域。

Header采用键值对的设计方式，具体格式规定为：

**message-header = field-name ":" [field-value]**

例如：“Host : example.com”。在每一个键值对后用CRLF结束行，进行下一个Header行的定义。

其中，具体的格式内容如下定义：

field-name = token

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

field-content = <the OCTETs making up the field-value

and consisting of either \*TEXT or combinations of token, separators, and

quoted-string>

## 5. 方法设计

request\_line中的请求方法部分应当包括GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT等，分别用以实现不同的请求功能。考虑到不同的功能，每一种方法如下设计：

1. GET：请求获取服务器上的相关资源。请求体中不包含请求数据，请求数据放在协议头。
2. POST：向服务器提交资源以待处理。所提交的资源应当放在请求体内部。
3. HEAD：用以检查资源或超链接的有效性以及可达性、获取头信息、检查网页是否被串改等；
4. PUT：请求服务器更新或插入资源；
5. DELETE：请求服务器删除某资源，具有破坏性；
6. CONNECT：通过服务器，请求其他网页的数据内容；
7. OPTIONS：获取http服务器支持的http请求方法，查看服务器的性能；
8. TRACE：回显服务器收到的请求，主要用于测试或诊断。

## 6.响应状态码设计

当客户端发送请求后，服务端应当进行相应的回应。这一回应是通过状态码来完成的。

状态码应当包括以下几类：

1. 1\*\*：信息，服务器收到请求，需要客户端继续进行信息的传递；

2. 2\*\*：操作被成功接收并处理；

3. 3\*\*：重定向，需要进一步操作并完成请求；

4. 4\*\*：客户端错误（语法错误或无法完成请求）；

5. 5\*\*：服务器错误（服务器在处理请求时发生了错误）。

# 协议实现

HTTP的采用**C语言**构建的工程进行实现，在ubuntu18.04环境下进行。其中，与协议词法、语法解析相关的内容在lexer.l、parser.y文件中进行定义。网络通信基于socket套接字完成。

源码依赖的类库主要包括有：unisted.h、socket.h、netdb.h、netinet/in.h、fcntl.h、netinet/ip.h以及parse.h。

源码的目录结构如下：

.

├── cp1

├── cp2

├── cp3

├── include

├── obj

└── src

其中具体包括的文件如下：

.

├── DockerFile

├── Makefile

├── README.md

├── cp1

│   ├── README

│   ├── cp1\_checker.py

│   ├── sample\_request\_example

│   └── sample\_request\_realistic

├── cp2

│   ├── README

│   ├── dumper.py

│   ├── liso.static\_site.tar.gz

│   ├── liso\_prototype.py

│   └── lowlevelhttptests.py

├── cp3

│   ├── README

│   ├── cgi.tar.gz

│   └── daemonize.c

├── echo\_client

├── echo\_server

├── example

├── include

│   ├── parse

│   └── parse.h

├── obj

│   ├── echo\_client.o

│   ├── echo\_server.o

│   ├── example.o

│   ├── lex.yy.o

│   ├── parse.o

│   └── y.tab.o

├── request

├── src

│   ├── echo\_client.c

│   ├── echo\_server.c

│   ├── example.c

│   ├── lex.yy.c

│   ├── lexer.l

│   ├── parse.c

│   ├── parser.y

│   ├── y.tab.c

│   └── y.tab.h

├── text.c

└── text.s

在src中包括有文件的所有源码，include文件夹中包含有句法解析器所引用的头文件。obj中包含有源码产生的链接文件，cpx中存放有测试工具。经过MakeFile后，将会在根目录下生成echo\_client与echo\_server的可执行文件，分别表示客户端与服务器端。

利用客户端向服务器端发送相应的请求后，服务器端将会返回一定的相应信息并打印在客户端的终端上。

# 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。

# 六、个人总结

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。