**《计算机网络》课程设计报告**





**第一周实验报告**

**学 号 3023005164 3023005041**

**姓 名 白孟轩 杨子豪**

**学 院 智能与计算学部**

**专 业 计算机科学与技术**

**年 级 23级**

**任课教师 赵增华**

**2025年 4月 2日**

# 一、报告摘要

根据给定的源代码进行HTTP协议的相关设计与实现，实现简单的echo功能，在完善相关模块的协议的设计与实现之中，更深入的理解http协议。

# 二、任务需求分析

首个阶段，我们要构建一个基础的echo网络服务器，处理POST、HEAD和GET请求。当服务器接收到来自客户端的含有多个请求行的消息时能够准确地解析这些信息，并发送相应的响应消息。处理逻辑分为三种主要情况：

**正确响应**：服务器接收到的是GET、HEAD或POST请求，它将这些请求信息重新封装后，再发送回客户端。

**未实现功能**：若收到的请求是GET、HEAD或POST之外的方法，或者其它非法的方法，服务器尚未实现这些功能，服务器应返回一个“HTTP/1.1 501 Not Implemented\r\n\r\n”的响应消息。

**格式问题**：如果客户端发送的请求名称正确而消息格式不正确，这一阶段的服务器可以识别这一错误，返回一个HTTP状态码为400的错误响应消息“HTTP/1.1 400 Bad request\r\n\r\n”。

此外，服务器还具备以下能力：

正确解析客户端发送的请求包，做出恰当的响应。

支持HTTP协议的并行请求处理。

能够同时处理来自多个客户端的并发连接。

# 三、协议设计

**3.1 总体设计**

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，是应用程序访问运输层的接口，在设计模式中把复杂的TCP/IP协议族隐藏在socket接口后面，socket负责组织数据来符合指定的协议，以实现向网络中的应用发送数据及接受其他应用发来的数据。详细见下文实现部分。

## **3.2 简单echo web server的设计**

1. 消息解析的具体实现：

1.echo\_server接受消息并返回结果的实现：

仔细观察echo\_server.c文件的代码，我们发现服务器原本是将接收到的消息直接转发给客户端。为了初步实现一些echo的功能，需要对调整这一行。当服务器接收到来自客户端的消息后，应首先将其解析到一个Request类型的指针中，然后根据以下三种情况分别处理：

如果http->method属性不属于GET、HEAD或POST中的任一种，服务器首先清空缓冲区，然后将字符串“HTTP/1.1 501 Not Implemented\r\n\r\n”写入缓冲区，并将这一消息发送回客户端。

如果解析过程中发现格式错误，即解析失败导致返回一个空指针，服务器首先清空缓冲区，然后将错误消息“HTTP/1.1 400 Bad request\r\n\r\n”写入缓冲区，并将这一消息发送给客户端。

如果Request中的http->method属性是GET、HEAD或POST中的任一种，并且可以正确解析，服务器应直接将接收到的请求echo给客户端。

处理完这些，我们发现并不能运行，问题出在了对parse函数的调用。我们尝试了各种方法来解决，例如修改头文件路径，但是依然不能完成调用。于是我们尝试自己书写接收和解析部分的代码，经过很多次尝试，最终才获得了80分。在和同学的一次讨论中，我们发现，除了修改echo\_server以外，还需要修改MAKEFILE文件，才能保证对parse的正确调用。修改完毕后，我们删除掉了冗余的代码，最终成功完成了所有评测数据，获得了100分。

## **3.3 基本HEAD、GET、POST方法的设计**

## **3.4 HTTP Pipelining的设计**

## **3.5 多个客户端的并发处理的设计**

**3.6 CGI的设计（选做）**

# 四、协议实现

## **4.1 简单echo web server的实现**

1. 消息解析的具体实现

在本次实验中，我们对HTTP消息解析进行了多项改进，包括多请求处理、扩展请求头容量以及彻底清理缓冲区。以下是具体实现过程的详细描述：

（1）实现多请求处理

通过分析parse.y文件代码发现，其原始设计仅能匹配单一的请求行和请求头，这与HTTP消息的实际规范存在偏差。为了解决这一问题，我们充分利用了Yacc和Lex工具支持的递归机制。具体而言，通过对变量request\_header进行递归定义，使得每次匹配request\_header时，不仅可以匹配单个请求头，还可以匹配由一个request\_header后跟request\_header\_all组成的结构。这样，我们就能够正确解析包含多个请求头的HTTP消息，从而实现了对多请求的有效处理。

（2）扩展请求头的容量

进一步分析parse.c文件后，我们发现当前设计中每个请求所分配的请求头数量固定为1，这种限制显然无法满足实际场景下的需求。为了解决这一问题，我们将每个请求可以容纳的请求头数量扩展至100。这一改动不仅提升了程序的灵活性，还显著增强了其应对复杂HTTP请求的能力。

（3）彻底清理缓冲区

在测试过程中，我们观察到以下异常现象：首先执行request\_get示例程序时结果正常；随后执行request\_400示例程序，结果同样正确；然而，当再次执行request\_get时，却得到了“HTTP/1.1 400 Bad request\r\n\r\n”的错误响应。经过逐行输出调试，我们发现问题出现在第三次解析消息时，得到的request指针为NULL。通过小组讨论，我们意识到，在处理完接收到的信息后，虽然已经对buff进行了memset操作并对request进行了free操作，但Lex和Yacc在词法分析和消息解析过程中仍然存在未被清空的隐藏缓冲区和栈。

为了解决这一问题，我们深入研究了源代码，并在lex.yy.c中找到了yylex\_destroy()函数。该函数能够彻底清除在词法分析和消息解析过程中残留的隐藏缓冲区和栈。通过调用此函数，我们成功解决了上述问题，确保了程序运行的稳定性与可靠性。

2. echo\_client文件输入的实现

在客户端请求的实现过程中，我们注意到echo\_client程序原本需要用户手动通过命令行输入请求消息。为了提升程序的灵活性和实用性，我们对其进行了改进，以支持从文件中读取输入数据。以下是具体实现过程：

改进需求分析：在使用echo\_client程序时，我们发现如果输入参数的数量不是3个，程序会直接报错并终止执行。然而，为了支持文件输入，我们需要将参数数量扩展至4个。因此，我们需要对代码进行调整，使程序在argc不等于4时能够输出错误信息并安全退出。

实现细节：在查看原代码中的信息读取部分时，我们注意到其采用了fgets()函数来获取用户输入的数据。为了实现从文件中读取输入，我们参考了example.c中的文件读取示例，采用了以下步骤：

使用open()函数打开指定的输入文件。

利用read()函数将文件内容读取至缓冲区中。

完成send()操作后，释放动态分配的内存空间，并关闭socket连接。

通过以上改动，我们成功实现了echo\_client程序从文件中读取输入的功能。具体实现细节可参考echo\_client.c文件。

## **4.2 基本HEAD、GET、POST方法的实现**

1. 获取URL的实现：

具体步骤如下：

1.比较HTTP URI：首先，将请求中的HTTP URI与空字符串进行比较。

2.不等：如果HTTP URI与空字符串不等，即存在URI，则继续执行下一步。

3.二者相等：如果HTTP URI与空字符串相等，即URI为空，则流程结束，可能需要返回一个错误或特定的响应。

4.将request->http\_uri的大小：如果URI不为空，接下来将请求中的HTTP URI的长度与当前系统所能容纳的最大URL长度进行比较。

5.小于等于：如果URI的长度小于或等于系统最大URL长度限制，则将该URI作为有效的URL继续处理。

6.大于：如果URI的长度超过了系统的最大URL长度限制，则流程结束，返回404错误，表示请求的资源未找到。

这个流程图描述了一个简单的URL有效性检查和长度限制的逻辑，确保了HTTP请求中的URI既不是空的，也没有超过系统设定的最大长度限制。如果URI不符合这些条件，流程将终止，并可能返回一个404错误。

其中URL\_MAX\_SIZE用来防止请求的URL过长导致缓冲区溢出，以实现合理管理缓冲区。

2. 服务器对读写磁盘文件异常处理的实现：

首先，调用stat()函数来检查文件是否存在。如果该函数返回-1，这表示文件不存在，程序应立即返回一个带有404错误代码的响应消息。随后，尝试使用open()函数来打开文件。如果得到的文件描述符fd\_in小于0，这表明打开文件的操作失败，同样需要返回一个带有404错误代码的响应消息。接下来，利用read()函数来读取文件内容。读取完成后，将RESPONSE\_200与文件内容结合，形成完整的响应消息，并存储在缓冲区中。最终，将缓冲区中的数据通过send()操作发送回客户端。

1. 四种请求异常的实现：

四种请求异常的返回信息分别保存在字符串数组：RESPONSE\_400、RESPONSE\_404、RESPONSE\_501与RESPONSE\_505中。

（1）400：如果解析失败即request指针为空时需要返回错误代码为400的响应信息，将RESPONSE\_400复制到缓冲区buf中，然后将buf发送回客户。

（2）404：在URL过大或者GET与HEAD方法中对磁盘文件的读写失败的时候错误代码为404的响应信息，将RESPONSE\_404复制到缓冲区buf中，然后将buf发送回客户。

（3）501：所请求的方法不是GET、HEAD和POST时需要返回错误代码为501的响应信息，将RESPONSE\_501复制到缓冲区buf中，然后将buf发送回客户。

（4）505：如果收到的request中的http\_version不是"HTTP/1.1"时需要返回错误代码为505的响应信息，将RESPONSE\_505复制到缓冲区buf中，然后将buf发送回客户。

4. GET、HEAD与POST方法的实现：

（1）GET：首先成功获取URL，然后再成功读取文件内容，然后使用strcat()函数将二者拼接到一起，然后将得到的新串放入缓冲区中然后返回。

我们需要获取URL，若失败，则直接返回404，若成功则获取文件状态并成功打开文件、读文件，若失败则返回404，若成功，将读到的内容与RESPONSE\_200拼接在一起然后返回

（2）HEAD：与GET方法相同，只是不用获取文件内容

我们需要获取URL，若失败，则直接返回404，若成功则获取文件状态并成功打开文件、读文件，若失败则返回404，若成功，将RESPONSE\_200放入缓冲区然后返回

（3）POST:直接echo回接收到的信息。

## **4.3 HTTP Pipelining的实现**

## **4.4 多个客户端的并发处理**

## **4.5 CGI的实现（**选做**）**

# 五、实验结果及分析

## 图片包含 背景图案 AI 生成的内容可能不正确。**5.1 简单echo web server的实验结果与分析**

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。文本

AI 生成的内容可能不正确。文本

AI 生成的内容可能不正确。通过本地测试可知：能够对GET、HEAD、POST方法相应；能够对没有实现的方法进行对应的处理；能够识别五种格式错误并返回相应的信息。成功实现了简单的echo web server。

## **5.2 HEAD、GET、POST方法的实验结果与分析**

详见实验图片通过本地测试可知：能够对GET、HEAD、POST方法相应；能够对没有实现的方法进行对应的处理；能够识别五种格式错误并返回相应的信息。成功实现了简单的echo web server

## **5.3 HTTP的并发请求的实验结果与分析**

## **5.4 多个客户端的并发处理的实验结果与分析**

**5.5 CGI的实验结果与分析（选做）**

# 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。