

天津大学

《计算机网络》课程设计 周进度报告

题目：第二周 实现 HEAD、GET 和 POST 方法

学 号: 3020202184 3020244344

姓 名: 刘锦帆 李镇州

学 院: 智能与计算学部

专 业: 计算机科学与技术

年 级: 2020 级

任课教师: 石高涛

2022 年 3 月 27 日

目 录

第一章 协议设计 ······	1
1.1 方法设计规则 ······	1
1.2 协议头部格式 ······	1
1.3 接受缓冲区设计 ······	2
1.4 日志记录模块设计 ······	2
第二章 协议实现 ······	3
2.1 方法实现 ······	3
2.1.1 错误码 ······	3
2.1.2 基本方法 ······	4
2.2 妥善管理缓冲区 ······	4
2.2.1 动态数组定义与实现 ······	4
2.2.2 动态数组使用 ······	5
2.3 日志记录模块实现 ······	5
2.4 读写磁盘文件错误处理 ······	5
第三章 实验结果及分析 ······	7
3.1 任务测试点 ······	7
3.2 结果分析 ······	7
3.3 日志 ······	7
第四章 进度总结及项目分工 ······	8
4.1 本周进度情况 ······	8
4.2 人员分工 ······	8

一 协议设计

1.1 方法设计规则

HTTP 1.1 三种基本方法的设计规则 (persistent connection) 和协议头部格式

GET

客户端请求指定资源信息，服务器返回指定资源。GET 是默认的 HTTP 请求方法，我们用 GET 方法来提交表单数据，然而用 GET 方法提交的表单数据只经过了简单的编码，同时它将作为 URL 的一部分向 Web 服务器发送，因此，如果使用 GET 方法来提交表单数据就存在着安全隐患上。GET 方法提交的数据是作为 URL 请求的一部分所以提交的数据量不能太大。

HEAD

获得报文头部信息，不回报文主体内容。用于确认 URI 的有效性及资源更新的日期时间等。HEAD 方法与 GET 类似，但是 HEAD 并不返回消息体。在一个 HEAD 请求的消息响应中，HTTP 报文中包含的元信息应该和一个 GET 请求的响应消息相同。这种方法可以用来获取请求中隐含的元信息，而无需传输实体本身。这个方法经常用来测试超链接的有效性，可用性和最近修改。一个 HEAD 请求响应可以被缓存，也就是说，响应中的信息可能用来更新之前缓存的实体。如果当前实体缓存实体阈值不同（可通过 Content-Length、Content-MD5、ETag 或 Last-Modified 的变化来表明），那么这个缓存被视为过期了。

POST

将客户端的数据提交到服务器，POST 方法是 GET 方法的一个替代方法，它主要是向 Web 服务器提交表单数据，尤其是大批量的数据。POST 方法克服了 GET 方法的一些缺点。通过 POST 方法提交表单数据时，数据不是作为 URL 请求的一部分而是作为标准数据传送给 Web 服务器，克服了 GET 方法中的信息无法保密和数据量太小的缺点。因此，出于安全的考虑以及对用户隐私的尊重，通常表单提交时采用 POST 方法。

1.2 协议头部格式

客户端向服务器发送一个请求，请求头包含请求的方法、URI、协议版本、以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。服务器以一个状态行作为响应，相应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。通常 HTTP 消息包括客户

机向服务器的请求消息和服务器向客户机的响应消息。这两种类型的消息由一个起始行，一个或者多个头域，一个只是头域结束的空行和可选的消息体组成。HTTP 的头域包括通用头，请求头，响应头和实体头四个部分。每个头域由一个域名，冒号和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

1.3 接受缓冲区设计

`mytcp_sockets_allocated` 是整个 `tcp` 协议中创建的 `socket` 的个数，由 `mytcp_prot` 的成员 `sockets_allocated` 指向。`mytcp_orphan_count` 表示整个 `tcp` 协议中待销毁的 `socket` 的个数，由 `mytcp_prot` 的成员 `orphan_count` 指向。`mysysctl_tcp_rmem` 表示接收缓冲区的大小限制，其缺省值分别是 4096bytes, 87380bytes, 174760bytes。它们可以通过 `/proc` 文件系统，在 `/proc/sys/net/ipv4/tcp_rmem` 中进行修改。`struct sock` 的成员 `sk_rcvbuf` 表示接收缓冲队列的大小，其初始值取 `mysysctl_tcp_rmem[1]`，成员 `sk_receive_queue` 是接收缓冲队列，结构跟 `sk_write_queue` 相同。`tcp socket` 的接收缓冲队列的大小既可以通过 `/proc` 文件系统进行修改，也可以通过 TCP 选项操作进行修改。套接字级别上的选项 `SO_RCVBUF` 可用于获取和修改接收缓冲队列的大小（即 `struct sock->sk_rcvbuf` 的值）。

1.4 日志记录模块设计

参考了 Apache 的标准，实现规范化输出，便于调试。

二 协议实现

第二周的协议实现主要分为“基本方法实现”、“缓冲区处理”、“日志记录模块”、“读写文件”和两个部分。

2.1 方法实现

服务器处理请求的方法被集中封装到文件 response.c 和 response.h 中。函数原型如图?? 所示。

具体而言，liso_server 首先使用 while 来接收信息，存储到 buffer 中（此处采用自定义的动态缓存，在下一节详细介绍）。然后通过一个 strstr() 函数获取确定一个完整的请求后，将它加载到动态缓存dbuf 中，并调用函数 handle_request 处理，返回的信息将继续通过dbuf 带出。由于测试集的要求，此处不需要实现 Persistent Connection 中对连接情况的判断。下面将分别介绍错误码和基本方法的实现。

2.1.1 错误码

400 400 错误是解析错误，由于之前在 Lab 1 已经实现了较为完备的 parse() 函数，于是在 Lab 2 中将不再 parse() 函数之内做修改。首先，通过图??左侧第 37 行调用 parse() 函数对请求进行解析，并通过第 40 行的判断确定一个 400 的错误，并随即调用函数 handle_400 处理之。然后，通过对自定义函数的调用，对动态缓冲数组dbuf 进行清空、加载返回信息等操作。

404 404 错误是指无法找到请求的文件，目前只会在 HEAD 和 GET 方法处理时出现。在处理 GET 和 HEAD 请求时，我们会通过 stat 等函数对要请求的文件进行预处理，此时即可判断该文件是否存在、是否可读等权限问题，然后决定是否返回 404 错误。

505 505 错误是指协议版本不支持，通过图?? 左侧第 53 行判断 my_http_version 与请求的 version 是否相同即可。

501 501 错误是指请求方法不支持，也可以通过一个 switch 判断出来。此处的 method_switch() 函数，将一个表示方法的字符串 ("GET", "POST", etc) 转化为图?? 右侧第 35 行定义的枚举类 METHOD。

2.1.2 基本方法

HEAD HEAD 方法的具体实现由函数 handle_head() 实现。首先，通过函数 strcmp() 判断请求路径是否为默认路径，如果是，则直接请求文件 index.html，否则请求路径添加到子文件夹"./static_site" 之下。然后，通过函数 stat() 判断是否为 404 错误。如果一切顺利，则通过在下一节介绍的动态缓冲区处理函数按照前文提到的格式，向返回值 dbuf 中添加一系列 response 和 headers。具体如图?? 所示。

a) Response.c 和 Response.h

b) liso_server.c 和 liso_client.c

图 2-1 核心代码

GET GET 方法在 HEAD 方法的基础之上，添加了读取文件的操作，根据模块化变成的思想，我们将其封装到函数 `get_file_content()` 之中。为了实现对缓冲区的管理、避免 Segment Fault，我们仍先通过 `stat` 函数对文件情况做初步确定，并先对动态数组进行扩容，再读入文件。文件读入部分将在后续小节进行介绍。

POST POST 方法的实现，根据实验要求，直接返回接收到的报文，于是在判断没有 400, 505，以及 501 错误之后，我们即可将携带请求信息的动态数组 `dbuf` 原路返回。

2.2 妥善管理缓冲区

由于 socket 编程本身的性质，在 `send` 函数中会主动的将信息拆分成合适大小再分别发送。所以我们只需要针对 **1.** 发送前，用户的缓冲区；**2.** 接收后集中存放的用户缓冲区；两个缓冲区进行管理。

2.2.1 动态数组定义与实现

我们的策略，是通过自定义的动态数组实现对一个动态长度的缓冲区的支持。具体定义及部分实现如图??所示。

部分结构体、函数介绍如下：

struct dynamic_buffer 该结构体定义了一个基地址 `*buf`，以及它的总大小 `capacity` 和当前大小 `current`。

init_dynamic_buffer() 该函数使用系统调用 `malloc` 将为一个新定义的动态数组分配由宏"DEFAULT_CAPACITY" 定义的空间。即对动态数组进行初始化。

append_dynamic_buffer() 该函数将一个字符串拼接到动态数组后面。首先判断当前空间是否够用，如果不够，则调用函数 `realloc` 为其增添空间，然后通过 `memcpy()` 函数，将新添加的字符串拼接到动态数组后面。

2.2.2 动态数组使用

动态数组的作用是解决缓冲区溢出问题，前文提到的两个可能出现溢出的地点，分别是读入文件时由于文件过大（或将其加入返回信息时）而溢出，另一个是在接收端虽然一次性接收的最大值由 `BUFSIZE` 控制，但是用户发送的请求大小不可定（例如 Lab3 中的 pipeline），导致缓冲区溢出。最终为了整体的和谐性，我们在大多数使用字符串数组的地方，都改用我们的动态数组。例如：

读入文件时 首先使用 `stat()` 获取文件大小，然后调用函数 `add_dynamic_buffer()` 函数对我们的动态数组进行可用空间的检验。最后再通过 `read()` 或者 `mmap()` 将文件写入动态数组。

Server 端 在 server 端，我们使用动态数组，存储即将返回给 client 的信息。由于 GET 可能会请求大小超标的文件，于是我们通过有保护机制的动态数组，安全地对它进行读取与返回。

接收 socket 信息时 由前文提到的 `recv()` 函数的性质，我们知道应该通过 `while` 来不断读取管道里的信息，如图?? 右侧第 171 行。通过 `append_dynamic_buffer()` 函数，将读取的信息添加到当前动态数组 `dbuf` 中。每次处理缓冲区，即是对 `dbuf` 中的数据进行拆分（确定一份请求报文）、解析与返回，最后调用 `update_dynamic_buffer()` 函数，将该报文从动态数组中丢弃。最大限度地节约空间，并解决缓冲区溢出问题。

验证

如图??，即使将缓冲区大小设置为 1，我们的 server 依旧正常处理并返回信息（测试样例为 pipeline，由于和 Persistent Connection 相关联，故一起实现了）。

2.3 日志记录模块实现

日志记录模块借用了 Apache 标准，使用 C 宏定义完成第一层封装，采用 `logger.c` 与 `logger.h` 配合，进行向特定文件的输入，实现了 `log` 的可跟踪性。具体宏定义如图?? 所示。

2.4 读写磁盘文件错误处理

只有 GET 和 HEAD 请求涉及对文件的处理。根据之前的描述，我们首先使用 `stat()` 函数对磁盘文件进行试探性的访问。如果在此时出错，可能有 1. 路径不存在；2. 没有查看权限。在服务器看来，我们只需返回 404 错误即可。

在 GET 的处理中，我们还需对文件进行读取，此时可能又会产生 3. 无法打开的错误；4. 文件为空；以及 5. 无法正确映射到内存；等问题（由于我们使用了上一节提到的动态数组，不会存在溢出问题）。

考虑完上述问题，我们实现了如图?? 的代码。

a) dynamic_buffer.c 和 dynamic_buffer.h

b) handle_head 函数

c) 日志文件

图 2-2 动态数组、日志文件以及 Handle Head 函数

a) buffer_test, BUF_SIZE=1

b) get_file_content 函数

图 2-3 测试 buffer 以及 get_file_content 函数

三 实验结果及分析

对于本次实验，我们使用了 API FOX 测试 GET, HEAD 以及 404 的返回（如图??）；使用浏览器测试服务器是否顺利返回文本、图片等（如图??）；使用 liso_client 测试新的缓冲区管理办法（如图??）。均通过。

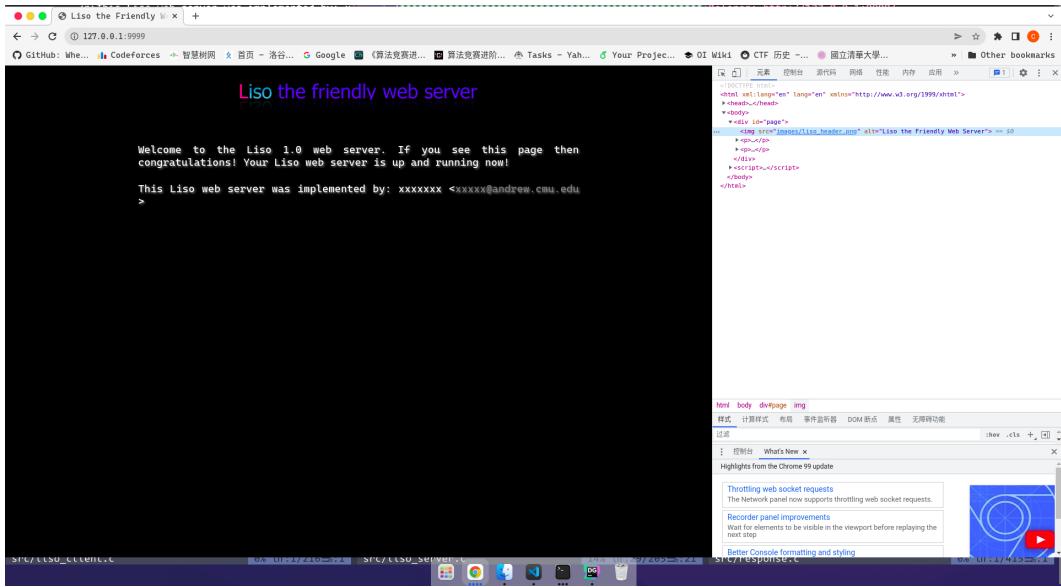


图 3-1 Liso Web Test

3.1 任务测试点

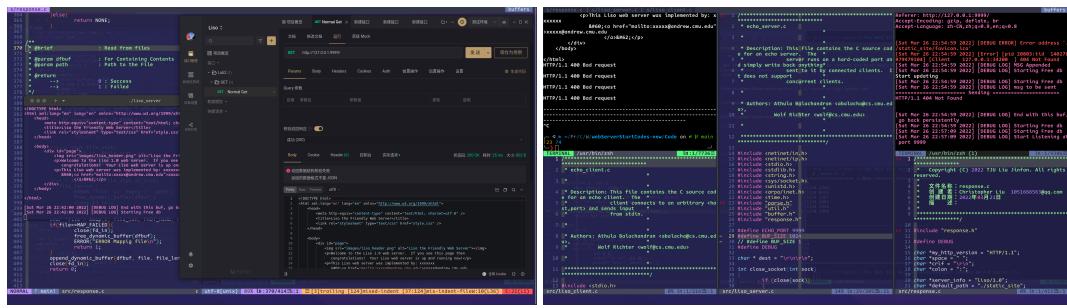
全部通过（如图??）。前几次因为没有正确修改文件名，所以一直"WA0"。直到问了助教才修改名字为 liso_server 以及 liso_client。

3.2 结果分析

在浏览器的测试中，发现和使用 liso_client 测试不同，当出现有 404 的情况时，我们的 liso_server 不会立刻进入下一轮监听，而是一直处于处理请求的情况。我们推测，是由于没实现 chunk 导致的。

3.3 日志

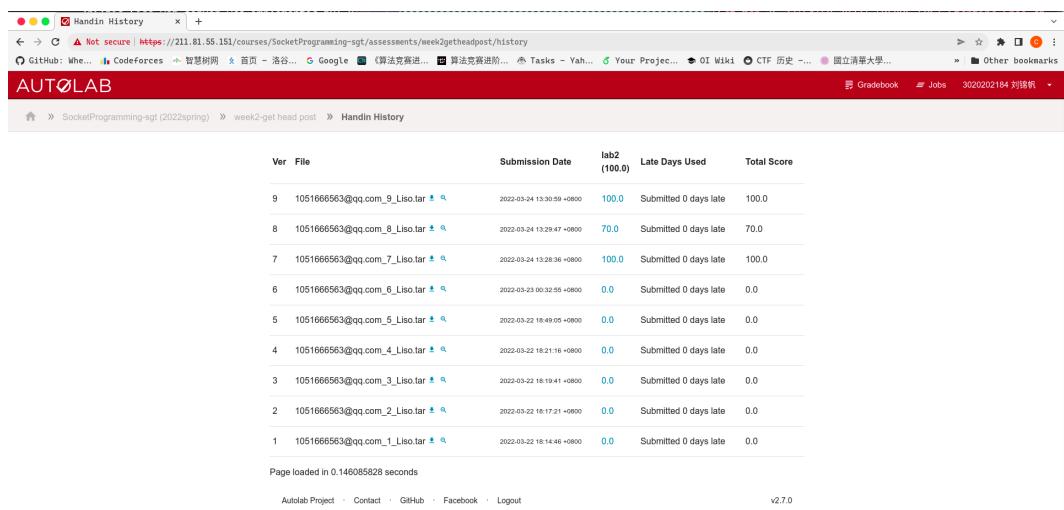
日志信息如图?? 所示。在日志信息里可以看到，我们的 recv 操作，每次最多接收 1025 字节的信息，单次可能不能完全接收一份报文的所有信息，但通过我们动态数组的管理，我们能够轻易完成 Pipeline 的测试。



a) Liso API FOX Test

b) Liso Test

c) 部分 log 输出



d) AutoLab 测试结果

图 3-2 测试结果展示

四 进度总结及项目分工

4.1 本周进度情况

本周主要完成了对 GET, HEAD, POST 以及出错情况的解析、分类和针对性返回。同时，通过 dynamic_buffer 对缓冲区以及发出、接收的报文进行封装，缓解了一部分可能出现的缓冲区溢出问题。在读写磁盘时，通过 stat 等函数进行预处理，避免了 segment fault 的发生。最后，创建了 logger.c 为格式化输出日志提供了接口。任务完成情况如表4-1所示。

	本周任务要求	完成	备注
1	完善服务器功能，能按照 RFC 2616 实现 HEAD、 GET 和 POST 的持久连接	✓	无
1.1	按照 RFC2616 响应 GET, HEAD 和 POST 方法	✓	无
1.2	支持 4 种 HTTP 1.1 出错代码: 400, 404, 501, 505	✓	无
1.3	妥善管理接收缓冲区，避免由于客户端请求消息 太常导致的缓冲区溢出问题	✓	无
2	服务器能处理读写磁盘文件时遇到的错误	✓	无
3	创建简化的日志记录模块，记录格式化日志	✓	无

表 4-1 本周进度完成表

4.2 人员分工

人员分工如表4-2所示。

人员	项目分工
刘锦帆	完成大部分代码工作，以及协议 实现部分
李镇州	完成 client 端的处理以及协议设 计部分的写作

表 4-2 人员分工表