天津大学

《计算机网络》课程设计
周进度报告

第一周 实现简单的 Echo Web Server

学　　号 **3020205015 3020210104**

姓　　名 石云天　姜卓雨

学　　院 未来技术学院

专　　业 计算机科学与技术

年　　级 **2020**

任课教师 周晓波

2023 年 6 月 1 日

目　录

一、协议设计 3
1.1 源文件架构 3
1.2 功能模块 3
1.2.1 分词模块 3
1.2.2 收发模块 4
1.3 消息解析方法 4
1.3.1 使用 lex 进行词法分析 4
1.3.2 使用 yacc 进行词法分析 5
1.4 任务目标 6
二、协议实现 7
2.1 完善语法分析器 7
2.2 完善 echo_server 7
2.3 其他问题 8
三、实验结果及分析 9
3.1 解析请求报文 9
3.2 响应 GET、HEAD、POST 信息 9
3.3 响应未实现方法 11
3.4 响应格式错误的方法 11
3.5 测试平台综合测试 12
四、进度总结 14

一、协议设计

1.1 源文件架构

```
project-1
├── DockerFile                  用于构建 docker 镜像
├── Makefile                    描述编译过程（make 指令调用）
├── README.md                   说明文档
├── cp1                         任务 1 测试文件
│   ├── README                  说明文档
│   ├── cp1_checker.py          任务 1 测试程序
│   ├── sample_request_example  只含一个 header 的 GET 请求
│   └── sample_request_realistic 含多个 header 的 GET 请求
├── cp2                         任务 2 测试文件
│   ├── README
│   ├── dumper.py
│   ├── liso.static_site.tar.gz
│   ├── liso_prototype.py
│   └── lowlevelhttptests.py
├── cp3                         任务 3 测试文件
│   ├── README
│   ├── cgi.tar.gz
│   └── daemonize.c
├── include                     自定义头文件的文件夹
│   └── parse.h                 语法分析程序的头文件
└── src                         源代码文件夹
    ├── echo_client.c           客户端程序
    ├── echo_server.c           服务端程序
    ├── example.c               样例程序
    ├── lexer.l                 词法分析器
    ├── parse.c                 语法分析程序
    └── parser.y                语法分析器
```

1.2 功能模块

1.2.1 分词模块

主要包括：include/parse.h src/parse.c src/lexer.l src/paeser.y。

首先 lex 根据 lexer.l 进行词法分析，然后 yacc 根据 parser.y 进行语法分析，最后 parse.c 将语法分析得到的结果放入结构体 Request 中。在本阶段任务中,需要完善语法分析器,使得服务端能够识别多个 header 的请求。

1.2.2 收发模块

主要包括：src/example.c src/echo_client.c src/echo_server.c。

以上源文件都将在编译后生成对应的二进制文件。程序 example 用于检测分词模块运行情况，将分词后的结果输出;程序 echo_client 为阶段任务 1 的客户端示例，读取标准输入的内容发送给服务端;程序 echo_server 为阶段任务 1 的服务器程序示例，用于实现对客户端发送的内容进行分析和响应。

由于 echo_server 为该阶段最主要的任务，所以需要对 echo_server.c 源文件进行详细分析。在给出的原始代码中，echo_server 实现了对报文的接收，并将接收到的报文原封不动的返回。程序首先初始化，创建、绑定和监听 socket。然后循环进行收发数据操作。在收发数据时，首先接受客户端的连接请求。然后持续不断地收发报文，将接收到的数据放在 buf 中，再将 buf 中的数据返回给客户端。直到客户端关闭连接或超时，然后服务端也断开连接。在以上过程中，如果发生错误，都将错误信息写入 strerr 并立刻退出程序。

在本阶段任务中，需要完善收发数据的操作，使得服务端能够识别不同的请求。

1.3 消息解析方法

1.3.1 使用 lex 进行词法分析

lex 的定义段规则如下表 1：

表 1 lex 段定义规则

正则表达式	说明	替代式
[0-9]	数字	digit
\x0d\x0a	回车换行	crlf
:	冒号	colon
\x20	空格	sp
[\x20\x09]*	空白	ws
\x0d\x0a(\x20|\x09)*	空白行	lws
[\(\)\<\>@,;:\\\”\/\[\]?=\{\}\x20\x09]	分隔符	separators
[\x0-\x1f\x7f]	控制符	ctl
[\x0-\x7f]{-}[\x0-\x1f\x7f]{-} [\{\}\(\)\<\>@\,;:\\\” /\[\]?=\x20\x09]	字符 （token 允许的）	token_char

lex 的规则段定义如下表 2，其中返回值对应 yacc 的标签：

表 2 lex 规则段定义

模式	说明	返回值
“\\”	反斜杠	t_backslash
“/\”	斜杠	t_slash
{crlf}	回车换行	t_crlf
{sp}	空格	t_sp
{ws}	空白	t_ws
{digit}	数字	t_digit
“.”	点	t_dot
{colon}	冒号	t_colon
{separators}	分隔符	t_separators
{token_char}	字符	t_token_char
{lws}	空白行	t_lws
{ctl}	控制符	t_ctl

1.3.2 使用 yacc 进行词法分析

yacc 的规则段定义如下表 3，其中部分标签来自 lex：

表 3 yacc 规则段定义

左式	说明	右式（以”|”分隔）
allowed_char_for_token	token 允许的字符	t_token_char|t_digit|t_dot
token	TOKEN	allowed_char_for_token|token allowed_char_for_token
allowed_char_for_text	text 允许的字符	allowed_char_for_token| t_separators|t_colon|t_slash
text	TEXT	allowed_char_for_text|text ows allowed_char_for_text
ows	空白	|t_sp|t_ws
request_line	请求行	token t_sp text t_sp text t_crlf
request_header	请求头	token ows t_colon ows text ows t_crlf
request_header_field	请求头域	request_header|request_header_field request_header
request	请求报文	request_line request_headder_field t_crlf

若最终匹配 request 成功，则说明该报文为请求报文，否则该报文无效。

我们利用 lex 和 yacc 进行词法分析：lex 词法分析器将输入解析成许多 token，yacc 语法分析器将 token 解析成具有一定优先级的语句进行计算，二者合作得到可执行文件进行编译。

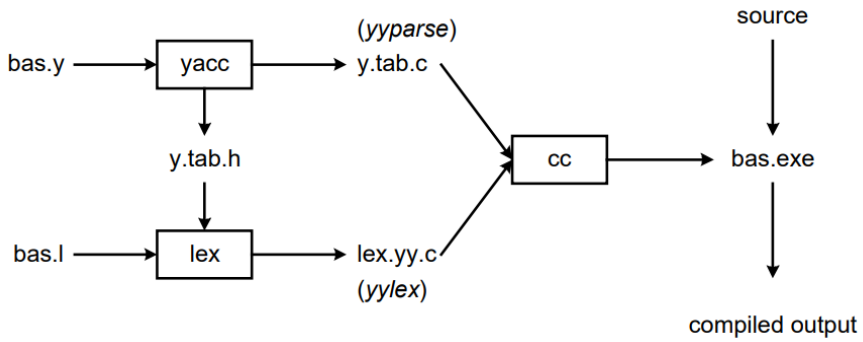

图 1 利用 lex/yacc 建立编译器

lexer.l 中先对词法做了一些定义,将他们分析成一个个 token。passer.y 中提供语法规则，而我们正是利用这些规则来判断 get 到的请求是否符合标准，而我们需要完善补充对于多行 request_header 的请求的解析。

具体的解析流程是服务器端收到消息后，调用 parse 函数进行解析，parse 函数中会先根据回车换行取出请求内容，然后分配空间并调用 yyparse 函数进行解析，并将解析的结果返回。服务器端根据返回的不同结果发送给客户端不同消息。

1.4 任务目标

基于本周的任务，第一个目标是能够识别多行 request_header，例如是 GET, HEAD 和 POST 方法，此时我们需要对于 parse.y 中的 request_header 语法规则作出修改，原代码在 parse.c 中只分配了一个 header 大小的内存空间，不能支持多 header 的 request 消息，现在要分配足够大的内存空间给 Request 结构体，同时还要修改 parse.y 中的规则来支持多请求行的 request 消息。

第二个目标是对于收到客户端发来的是除 GET, HEAD 和 POST 以外的其它 method，返回响应消息“HTTP/1.1 501 Not Implemented\r\n\r\n”。这里需要在 echo_server.c 中对收到 client 端 send 来的 request 消息进行判断，如果是上述三种 method 则 echo 回去，如果不是则向 client 端 send"HTTP/1.1 501 Not Implemented\r\n\r\n"。

第三个目标是识别格式错误的消息，并返回 400 错误代码。添加当消息格式错误时解析函数会返回 NULL，所以在 echo_server.c 的对 client 端发送来的 request 消息进行判断时加入 parse()返回 NULL 的情况，在 NULL 时向 client 端

send“HTTP/1.1 400 Bad request\r\n\r\n”。

二、协议实现

2.1 完善语法分析器

由于初始的程序仅能识别一个 header 行的请求报文，而在实际的 HTTP 协议中，可以有多个 header，因此要修改 parser.y 文件，使得语法分析器支持识别多个 header 行。

因此定义规则:左式为 request_header_field，右式 1 为 request_header，右式 2 为request_header_line request_header。通过这个规则，就可以将多个request_header 规约为 request_header_field。然后将 request 的规约规则改为 request_line request_header_field t_crlf，这样就可以实现对多个 header 的请求报文的识别。

然而，如果就这样运行 example 程序测试 sample_request_realistic，会产生运行时错误，从而未能分析完请求报文就强制退出程序。于是通过 gdb 分析问题，发现在运行时出现了 segmentation fault，通过分析问题出现的位置可以确定，当匹配 request_header 时，会将 header 的内容写入 parsing_request->headers。但是初始时只给 parsing_request->headers 分配了一个 header 的空间，导致遇到多个 header 时，分配的空间不足以放置多余的 header。所以可以在每次匹配 request_header_field 时，重新分配空间，由于 parsing_request->header_count 记录当前已匹配的 header 数，所以可以通过公式:

$$parsin\_request{-}> headers = realloc(parsin\_request{-}> headers, sizeof(Request\_header) * (\ parsin\_request{-}> header\_count + 1));$$

来重新分配空间。将这个语句写入 request_header_field 两个右式的动作即可。

2.2 完善 echo_server

初始的 echo_server 仅能将收到的报文原封不动地返回，无法对报文进行分析处理，因此需要修改 echo_server.c 文件，对不同的报文做不同的处理。

由于需要调用分词模块，所以程序要#include"parse.h”。然后，在 while((readret = recv(client_sock， buf，BUF_SIZE，0)) >= 1)循环中，需要调用 parse 函数对请求报文进行分词处理。Parse 函数将返回一个指向 Request 结构体类型的指针,该结构体中存放从请求报文中解析得到的数据。如果解析失败，则返回一个 NULL 指针。

得到指向 Request 结构体类型的指针后,首先判断该指针是否为空。若为空，

则说明该报文无法分析识别，需要向客户端返回"HTTP/1.1 400 BadRequestlrln\rin"，因此清除 buf 的内容，并向 buf 写入上述字符串。若指针不为空，则检查结构体中 http_method 的内容，如果不为 GET、HEAD、POST 请求，则需要向客户端返回"HTTP/1.1 501 Not Implementedrln\rin"，因此清除 buf 的内容，并向 buf 写入上述字符串。如果是 GET、HEAD、POST 请求，保持 buf 原封不动即可。然后再向客户端发送 buf 中的内容。

由于 Request 结构体的内存空间是通过 malloc 分配的，因此为防止内存泄漏，需要释放空间。所以若指向 Request 结构体类型的指针不为 NULL，则需要 frec 掉其对应的内存空间。但如果指针为空，结构体也分配了空间，但 echo_server.c 中无法获取到该结构体的地址，因此该空间只能在 parse.c 中释放。于是在 parse.c 中，如果语法分析不成功，则立刻 free 掉分配给 Request 结构体的内存空间。

此时，单线程的 echo_server 便完成了。但此时如果使用 make 进行编译，会提示找不到 parse，这是因为在编译时未连接 parse.o 文件。修改 makefile 文件，编译时将所有依赖加入编译指令即可。

使用 cpl_checker.py 对程序进行测试。由于只实现了单线程，所以使用的测试指令为 python cp1/cp1_checker.py 127.0.0.1 9999 100 1 1（使用 python 运行 cp1/cpl_checker.py，服务端 IP 地址为 localhost(127.0.0.1)，端口号为 9999，测试 100 个请求，每个请求读写次数为 1 次，建立 1 个连接)。但是，测试结果总是失败。分析 echo_server 在标准输出中输出的内容发现 yacc 每次规约失败后，都将从上次失败的地方重新规约,导致只要规约失败一次,后续都将会规约失败。因此在调用 yacc 前，需要设置 yacc 从头开始规约。通过在互联网上的查找，找到了 yyrestart(NULL);语句可以使得 yacc 每次从头开始规约。因此在 parse.c 文件中，调用 yyparse 函数前，加入上述语句，即可解决问题。

2.3 其他问题

由于 echo_server 不会自动结束，会循环执行 while (1)内的代码，只有使用 Ctrl+C 才能强制终止程序。而强制终止程序就不会主动解除 socket 的绑定，导致短时间内再次运行程序时会出现无法绑定 socket 的情况。这时，只能等待 socket 超时才能重新运行 echo_server。为了避免这个问题，需要截获 Ctrl+C。因此设定一个全局变量 app_stopped，初始置为 0,当识别到 Ctrl+C 按下时,将 app_stopped 置为 1。每次 while 循环，都将检测 app_stopped 的值，当检测到 app_stopped 为 1 时，断开 socket 连接并退出循环。同时，给 socket 应用 SO_REUSEADDR 选项也能解决由于其他情况导致的 socket 端口被占用的问题，这样就可以解决无法

绑定 socket 的问题。

三、实验结果及分析

3.1 解析请求报文

Client 端发送请求报文，观察服务器端的解析情况：

```
----- Echo Server -----
t:token_char G
t:token_char E
t:token_char T
t:sp ' ';
t:slash;
t:sp ' ';
t:token_char H
t:token_char T
t:token_char T
t:token_char P
t:slash;
t:digit 1;
t:dot;
t:digit 1;
t:crlf;
t:token_char H
t:token_char o
t:token_char s
t:token_char t
t:colon;
t:sp ' ';
t:token_char w
t:token_char w
t:token_char w
t:dot;
t:token_char c
t:token_char s
t:dot;
```

图 2 测试结果

发现服务器端能够正确解析报文的内容并打印。

3.2 响应 GET、HEAD、POST 信息

发送下图所示的请求信息，观察 client 端收到的反馈情况：

```
1  GET / HTTP/1.1
2  Host: www.cs.cmu.edu
3  Connection: keep-alive
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
6  Accept-Encoding: gzip, deflate, sdch
7  Accept-Language: en-US,en;q=0.8
```

图 3 client 端反馈

```
root@8e3696f32204:/home/project-1# ./echo_client localhost 9999 samples/r
equest_get
Sending GET / HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8


Received GET / HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

图 4 测试结果

发现 get 请求头的报文直接返回了原始的报文内容，符合任务要求

然后发送 head 请求头的报文进一步测试：

```
HEAD / HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

图 5 head 报文测试

```
root@8e3696f32204:/home/project-1# ./echo_client localhost 9999 samples/r
equest_head
Sending HEAD / HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8


Received HEAD / HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

图 6 测试结果

发现 head 请求头的报文也直接返回了原始的报文内容，符合任务要求。

3.3 响应未实现方法

发送除 head，get，post 外其他方法的请求信息，服务器需要正确返回相应的响应信息：

```
DELETE /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

图 7 未实现方法测试

```
root@8e3696f32204:/home/project-1# ./echo_client localhost 9999 samples/r
equest_delete
Sending DELETE /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8


Received HTTP/1.1 501 Not Implemented
```

图 8 测试结果

发现服务器返回了预期的响应信息。

3.4 响应格式错误的方法

修改格式使其格式错误：

```
DELETE
/~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

图 9 格式错误

```
root@8e3696f32204:/home/project-1# ./echo_client localhost 9999 samples/r
equest_delete
Sending DELETE
/~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/39.0.2171.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8


Received HTTP/1.1 400 Bad request
```

图 10 测试结果

发现服务器能够识别客户端消息的格式错误，并且返回错误代码的响应信息。

3.5 测试平台综合测试

使用测试平台测试，输出如下：

```
Autograder [Thu Jun  1 17:06:36 2023]: Received job
SocketProgramming2023Spring_week1echoserver_1_yuntian_shi5015@tju.edu.c
n:581
Autograder [Thu Jun  1 17:06:42 2023]: Success: Autodriver returned
normally
Autograder [Thu Jun  1 17:06:42 2023]: Here is the output from the
autograder:
---
Autodriver: Job exited with status 0
tar xvf autograde.tar
Liso-handout/
Liso-handout/cp1_checker.py
Liso-handout/driver.sh
cp -r Liso.tar Liso-handout
(cd Liso-handout; tar xvf Liso.tar; ./driver.sh)
cgi/
cgi/cgi.tar.gz
cgi/daemonize.c
cgi/README
cp1/
cp1/cp1_checker.py
DockerFile
echo_client
echo_server
example
include/
include/parse.h
Makefile
obj/
obj/echo_client.o
obj/echo_server.o
obj/example.o
obj/lex.yy.o
obj/parse.o
obj/y.tab.o
README.md
samples/
```

```
samples/request_delete
samples/request_get
samples/request_head
samples/request_pipeline
samples/request_post
samples/sample_request_example
samples/sample_request_realistic
src/
src/echo_client.c
src/echo_server.c
src/example.c
src/lex.yy.c
src/lexer.l
src/parse.c
src/parser.y
src/y.tab.c
src/y.tab.h
static_site/
static_site/images/
static_site/images/liso_header.png
static_site/index.html
static_site/style.css
Compiling
make[1]: Entering directory '/home/autograde/autolab/Liso-handout'
rm -f obj/y.tab.o obj/lex.yy.o obj/parse.o obj/example.o example
echo_server echo_client src/lex.yy.c src/y.tab.*
rm -f -r obj
make[1]: Leaving directory '/home/autograde/autolab/Liso-handout'
make[1]: Entering directory '/home/autograde/autolab/Liso-handout'
yacc -d src/parser.y
src/parser.y: warning: 6 shift/reduce conflicts [-Wconflicts-sr]
mv y.tab.c src/y.tab.c
mv y.tab.h src/y.tab.h
mkdir obj
gcc -Iinclude -g -Wall -c src/y.tab.c -o obj/y.tab.o
flex -o src/lex.yy.c src/lexer.l
gcc -Iinclude -g -Wall -c src/lex.yy.c -o obj/lex.yy.o
src/lex.yy.c:1387:16: warning: ‘input’ defined but not used [-Wunused-
function]
     static int input  (void)
```

```
                ^~~~~
src/lex.yy.c:1344:17: warning: ‘yyunput’ defined but not used [-
Wunused-function]
     static void yyunput (int c, char * yy_bp )
                 ^~~~~~~
gcc -Iinclude -g -Wall -c src/parse.c -o obj/parse.o
src/parse.c: In function ‘parse’:
src/parse.c:56:3: warning: implicit declaration of function ‘yyrestart’
[-Wimplicit-function-declaration]
   yyrestart(NULL);
   ^~~~~~~~~
gcc -Iinclude -g -Wall -c src/example.c -o obj/example.o
gcc obj/y.tab.o obj/lex.yy.o obj/parse.o obj/example.o -o example
gcc -Iinclude -g -Wall -c src/echo_server.c -o obj/echo_server.o
gcc -Werror obj/echo_server.o obj/parse.o obj/y.tab.o obj/lex.yy.o -o
echo_server
gcc -Iinclude -g -Wall -c src/echo_client.c -o obj/echo_client.o
gcc -Werror obj/echo_client.o -o echo_client
make[1]: Leaving directory '/home/autograde/autolab/Liso-handout'
Running
{"scores": {"lab1": 100.00}}

Score for this problem: 100.0

Graded by:
```

lab1 (100.0)	Late Days Used	Total Score
100.0	Submitted 0 days late	100.0

四、进度总结

表 4 本周任务完成表

本阶段任务要求	完成	未完成	备注
1. 阅读 HTTP/1.1 的标准文档 RFC2616	√		
2. 搭建编程环境	√		
3. 熟悉 socket 编程方法	√		
4. 掌握 lex 和 yacc 正确解析消息	√		

（message）的方法			
5.1 实现简单的 echo web server，Echo Get，HEAD，POST	√		
5.2 响应没有实现的方法	√		
5.3 相应错误的方法	√		
6. 功能测试	√		

表 5 上周任务改进表

序号	上周任务	改进内容	备注
1	无		