**《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](#_Toc136640772)** [3](#_Toc136640772)

[1.1 源文件架构 3](#_Toc136640773)

[1.2 功能模块 3](#_Toc136640774)

[1.2.1 分词模块 3](#_Toc136640775)

[1.2.2 收发模块 4](#_Toc136640776)

[1.3 消息解析方法 4](#_Toc136640777)

[1.3.1 使用lex进行词法分析 4](#_Toc136640778)

[1.3.2 使用yacc进行词法分析 5](#_Toc136640779)

[1.4 任务目标 6](#_Toc136640780)

[**二、协议实现** 7](#_Toc136640781)

[2.1 完善语法分析器 7](#_Toc136640782)

[2.2 完善echo\_server 7](#_Toc136640783)

[2.3 其他问题 8](#_Toc136640784)

[**三、实验结果及分析** 9](#_Toc136640785)

[3.1 解析请求报文 9](#_Toc136640786)

[3.2 响应GET、HEAD、POST信息 9](#_Toc136640787)

[3.3 响应未实现方法 11](#_Toc136640788)

[3.4 响应格式错误的方法 11](#_Toc136640789)

[3.5 测试平台综合测试 12](#_Toc136640790)

[**四、进度总结** 14](#_Toc136640791)

**一、协议设计**

## 1.1 源文件架构



## 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.1 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。

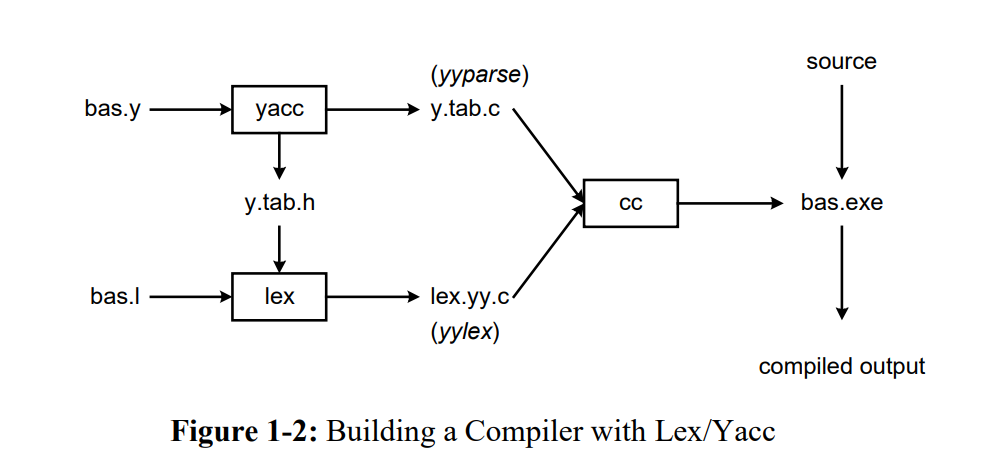


图1 利用lex/yacc建立编译器

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:

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：

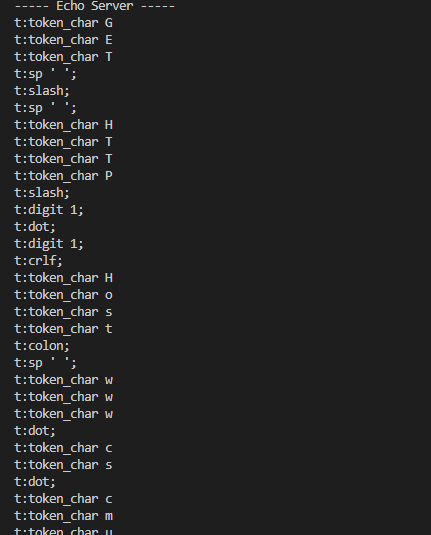


图2 测试结果

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：

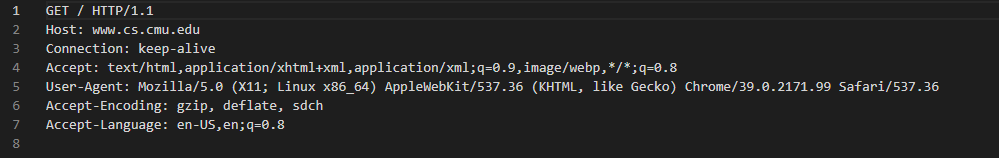


图3 client端反馈

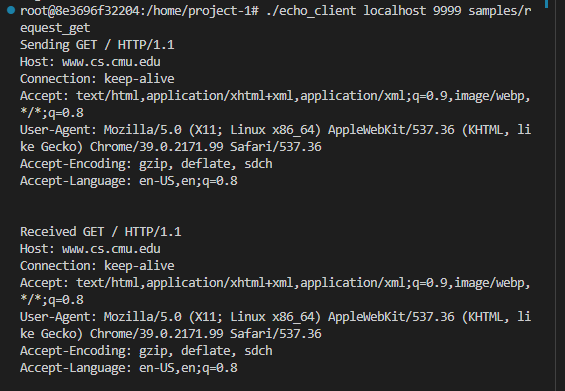


图4 测试结果

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：

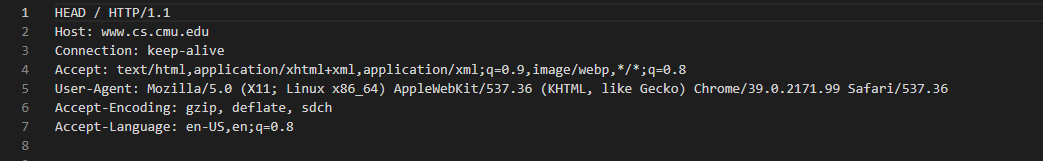


图5 head报文测试

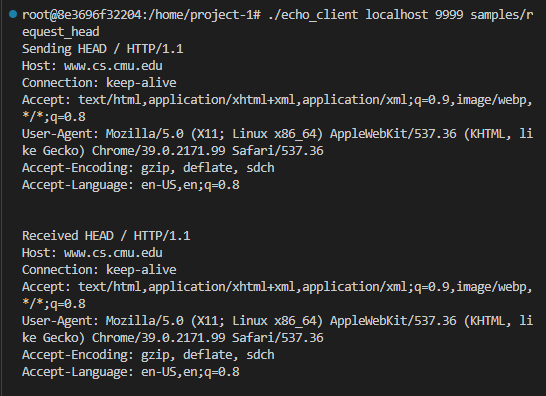


图6 测试结果

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：

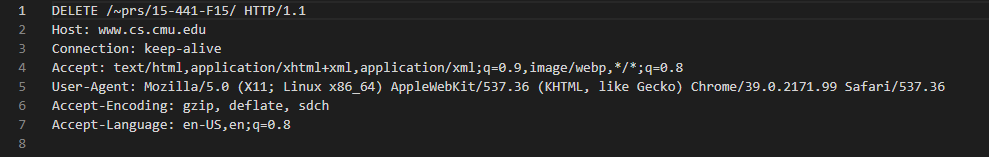


图7 未实现方法测试

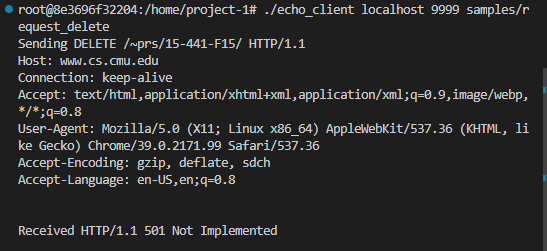


图8 测试结果

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：

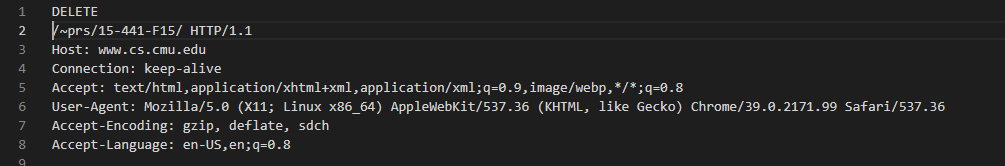


图9 格式错误

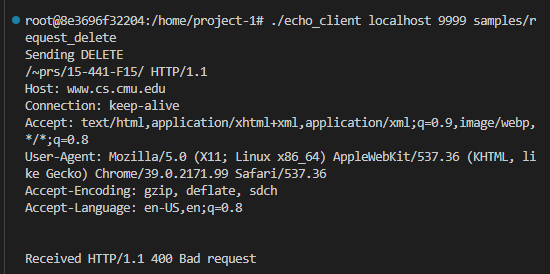


图10 测试结果

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.cn: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

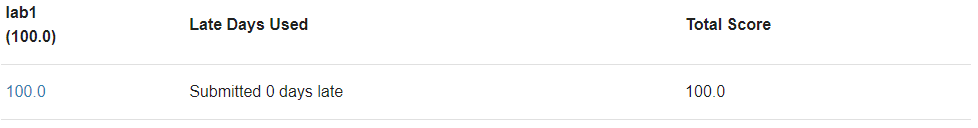
make[1]: Leaving directory '/home/autograde/autolab/Liso-handout'

Running

{"scores": {"lab1": 100.00}}

Score for this problem: 100.0

Graded by:



**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 |  |  |