

HTTP服务器/客户端实验

2020年9月29日

蔡润泽

本实验 [Github](#) [地址](#)

实验内容

1、使用C语言分别实现最简单的HTTP服务器和HTTP客户端

- 服务器监听 80 端口，收到HTTP请求，解析请求内容，回复HTTP应答
- 对于本地存在的文件，返回 HTTP 200 OK 和相应文件
- 对于本地不存在的文件，返回 HTTP 404 File Not Found

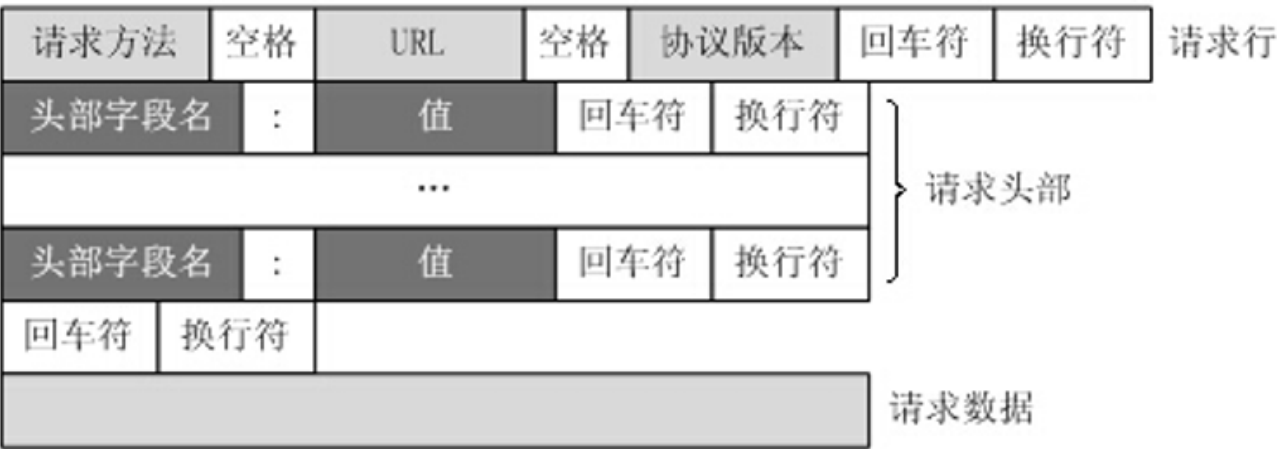
2、服务器、客户端需要支持HTTP Get方法

3、服务器使用多线程支持多路并发

设计思路

1、已有代码框架已经能实现socket下的echo，而实现HTTP client和server首先要做的是实现HTTP GET请求与相应。

2、在查询 HTTP GET 请求格式后，了解到 GET 请求报文格式如下图：



因此，本设计在 http-client.c 文件里添加了字符串 request_head 来生成请求报文，并在程序运行时通过 scanf 获得用户输入的请求文件路径。在加上请求头部后，Client 将请求发送给 Server。

3、Server 在收到 HTTP GET 请求格式后，了解到响应报文格式如下图：



因此，本设计在 http-server.c 文件里添加了 int msg_handler(char * msg, char * path) 函数来解析请求报文，并提取出请求文件的地址。

4、Server 在打开目标文件后，读取目标文件的内容，并将内容传递至 return_message，return_message 此外还会附上必要的响应头部属性最后 Server 会将 return_message 发送给 Client。

5、Server 为实现多线程发送文件的功能，本设计增加了支持 pthread 相关函数。通过 pthread_detach 请求处理线程来实现多路并发。相关代码如下：

```
while(1) {
    pthread_t thread;
    pthread_create(&thread, NULL, handle_request, &s);
    pthread_detach(thread);
}
```

测试项目

1、所实现的HTTP服务器、客户端可以完成上述功能

使用客户端连续多次获取文件以及请求不存在的文件

本客户端通过与 SimpleHTTPServer 建立连接来测试客户端连续多次获得文件的能力，如下图：

Capturing from h1-eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ...<Ctrl-/>

Expression...

No.	Time	Source	Destination	Protocol	Length	Info
68	68.695308628	52:05:a4:3a:84:c7	0e:48:d9:3e:ba:42	ARP	42	10.0.0.1 is at 52:05:a4:3a:84:c7
69	68.695383605	0e:48:d9:3e:ba:42	52:05:a4:3a:84:c7	ARP	42	10.0.0.2 is at 0e:48:d9:3e:ba:42
70	99.927163452	fe80::10cf:d7ff:fe...	ff02::2	ICMPv6	70	Router Solicitation
71	112.215809766	fe80::5005:a4ff:fe3...	ff02::2	ICMPv6	70	Router Solicitation
72	112.216325647	fe80::c48:d9ff:fe3e...	ff02::2	ICMPv6	70	Router Solicitation
73	114.264220523	fe80::940f:f9ff:fe...	ff02::2	ICMPv6	70	Router Solicitation
74	116.536751155	fe80::10cf:d7ff:fe...	ff02::fb	NDMS	203	Standard query (
75	138.775600535	10.0.0.2	10.0.0.1	HTTP	169	GET /3.txt HTTP/1.0
76	138.775616427	10.0.0.1	10.0.0.2	TCP	66	80 -> 42352 [ACK
77	138.776064420	10.0.0.1	10.0.0.2	TCP	83	80 -> 42352 [PSH
78	138.776076902	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
79	138.776097217	10.0.0.1	10.0.0.2	TCP	104	80 -> 42352 [PSH
80	138.776115751	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
81	138.776147610	10.0.0.1	10.0.0.2	TCP	103	80 -> 42352 [PSH
82	138.776153116	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
83	138.776165591	10.0.0.1	10.0.0.2	TCP	92	80 -> 42352 [PSH
84	138.776169692	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
85	138.776186293	10.0.0.1	10.0.0.2	TCP	86	80 -> 42352 [PSH
86	138.776190604	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
87	138.776207862	10.0.0.1	10.0.0.2	TCP	112	80 -> 42352 [PSH
88	138.776212396	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
89	138.776223431	10.0.0.1	10.0.0.2	TCP	68	80 -> 42352 [PSH
90	138.776227397	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
91	138.776271229	10.0.0.1	10.0.0.2	HTTP	80	HTTP/1.0 200 OK
92	138.776278041	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [ACK
93	138.776343091	10.0.0.1	10.0.0.2	TCP	66	80 -> 42352 [FIN
94	138.776557822	10.0.0.1	10.0.0.2	TCP	66	42352 -> 80 [FIN
95	138.776565038	10.0.0.1	10.0.0.2	TCP	66	80 -> 42352 [ACK
96	138.776616699	10.0.0.1	10.0.0.2	TCP	74	42354 -> 80 [SYN
97	138.776626957	10.0.0.1	10.0.0.2	TCP	74	80 -> 42354 [SYN
98	138.776636819	10.0.0.1	10.0.0.2	TCP	66	42354 -> 80 [ACK
99	143.960776674	0e:48:d9:3e:ba:42	52:05:a4:3a:84:c7	ARP	42	Who has 10.0.0.1
100	143.960787842	52:05:a4:3a:84:c7	0e:48:d9:3e:ba:42	ARP	42	10.0.0.1 is at 52:05:a4:3a:84:c7
101	143.963399517	52:05:a4:3a:84:c7	0e:48:d9:3e:ba:42	ARP	42	Who has 10.0.0.2
102	143.963418555	0e:48:d9:3e:ba:42	52:05:a4:3a:84:c7	ARP	42	10.0.0.2 is at 0e:48:d9:3e:ba:42

"Node: h1"

root@ubuntu:~/code/example# python -m SimpleHTTPServer 80 & wireshark

[1] 4866

Serving HTTP on 0.0.0.0 port 80 ...

error: XDG_RUNTIME_DIR not set in the environment.

10.0.0.2 - - [29/Sep/2020 19:17:18] "GET /1.txt HTTP/1.1" 200 -

10.0.0.2 - - [29/Sep/2020 19:18:19] code 404, message File not found

10.0.0.2 - - [29/Sep/2020 19:18:19] "GET /2.txt HTTP/1.1" 404 -

10.0.0.2 - - [29/Sep/2020 19:19:34] "GET /3.txt HTTP/1.1" 200 -

"Node: h2"

len : 202

server reply : HTTP/1.0 200 OK

Server: SimpleHTTP/0.6 Python/2.7.12

Date: Tue, 29 Sep 2020 11:17:18 GMT

Content-type: text/plain

Content-Length: 16

Last-Modified: Sat, 26 Sep 2020 06:09:34 GMT

crz is great 1

HTTP/1.0 200 OK

enter message : 2.txt

send msg:

GET /2.txt HTTP/1.1

Accept: */*

Accept-Encoding: identity

Host: 10.0.0.1

Connection: Keep-Alive

len : 148

server reply : HTTP/1.0 404 File not found

Server: SimpleHTTP/0.6 Python/2.7.12

Date: Tue, 29 Sep 2020 11:18:19 GMT

Connection: close

Content-Type: text/html

save failed

HTTP/1.0 404 File not found

enter message : 3.txt

send msg:

GET /3.txt HTTP/1.1

Accept: */*

Accept-Encoding: identity

Host: 10.0.0.1

Connection: Keep-Alive

len : 200

server reply : HTTP/1.0 200 OK

Server: SimpleHTTP/0.6 Python/2.7.12

Date: Tue, 29 Sep 2020 11:19:34 GMT

Content-type: text/plain

Content-Length: 14

Last-Modified: Sat, 26 Sep 2020 05:13:57 GMT

crz is great

HTTP/1.0 200 OK

enter message :

左侧 Xterm 界面显示了 Client 接收文件的反馈。Client 在控制台打印输出了请求的内容和相应的内容。

左侧 Xterm 里，Client 分别请求了 1.txt，2.txt 以及 3.txt。

1.txt 里的内容为 crz is great 1，2.txt 不存在，3.txt 里的内容为 crz is great。左边输出内容代表三个请求均反馈正确，且正常收取了 1.txt 和 3.txt 两个文件。接收的文件名在原文件名前增加 recv_，如下图：



同时启动多个客户端进程分别获取文件

通过 pthread 支持多线程多路并发，本设计支持同时启动多个客户端进程分别获取文件，具体的测试实现是借助 wget 10.0.0.1/1.txt & wget 10.0.0.1/3.txt 实现两个 wget 指令同时请求，测试结果如下：

```
--2020-09-29 20:05:59-- http://10.0.0.1/1.txt
Connecting to 10.0.0.1:80... connected.
--2020-09-29 20:05:59-- http://10.0.0.1/3.txt
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... HTTP request sent, awaiting response...
200 OK
200 OK
Length: 14Length:
16
Saving to: '3.txt.1'

Saving to: '1.txt.3'

1.txt.3          0%[                  ]    0  --.-KB/s    in 0s
1.txt.3          100%[=====>]    16  --.-KB/s    in 0s

2020-09-29 20:05:59 (1.19 MB/s) - '1.txt.3' saved [16/16]

3.txt.1          100%[=====>]    14  --.-KB/s    in 0s

2020-09-29 20:05:59 (1.32 MB/s) - '3.txt.1' saved [14/14]

[1]+  Done                  wget 10.0.0.1/1.txt
root@ubuntu:~/code/example#
```

由图可知， `1.txt` 和 `3.txt` 均被顺利接收。

2、使用”python -m SimpleHTTPServer 80”和wget分别替代服务器和客户端，对端可以正常工作

上述测试方式即可验证每一端均可正常运行。另外为了测试本设计中的 `Client` 和 `Server` 之间可以正常通行，增加了下述测试：

即多个 `Client` 向 `Server` 发送请求。结果如下：

```
Accept: */*
Accept-Encoding: identity
Host: 10.0.0.1
Connection: Keep-Alive
```

```
path: 1.txt
path: 3.txt
HTTP/1.1 200 OK
Content-Length: 16
Connection: Keep-Alive
```

```
crz is great 1
```

```
HTTP/1.1 200 OK
Content-Length: 14
Connection: Keep-Alive
```

```
crz is great
```

遇到的问题

在实验过程中，通过抓包发现，Client 向 SimpleHTTPServer 发送请求信号后，如果多次发送请求，SimpleHTTPServer 在第一次正常返回响应后，TCP会发出RST信号，导致 Client 如果不重新建立连接就会发生异常。

因此本设计中，如果 Client 向 SimpleHTTPServer 连续发包的话，需要在每次发包后重连。实现代码及其注释如下：

```
/* The following part used depends on whether uses SimpleHttpServer.
If you want to use SimpleHttpServer, you should enable this part
because SimpleHttpServer will send TCP RST signal after once connection.
*/

/*
close(sock);
sock = socket(AF_INET, SOCK_STREAM, 0);
connect(sock, (struct sockaddr *)&server, sizeof(server));
*/
```

参考文献及网站

[1] <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

[2] <https://www.cnblogs.com/an-wen/p/11180076.html>