

# Socket 实验设计报告

中国科学院大学

页

2022 年 4 月 5 日

## 一、HTTP 服务器实验

### 1. 实验内容

(1) 使用 C 语言分别实现最简单的 HTTP 服务器：

①服务器同时支持监听 HTTP (80 端口) 和 HTTPS (443 端口)，收到 HTTP 请求，解析请求内容，回复 HTTP 应答。443 端口需要支持 200 OK、206 Partial Content 和 404 Not Found 等 3 个状态码，80 端口需要支持 301 Moved Rermanently 状态码。

②服务器需要支持 HTTP Get 方法。

③服务器创建两个线程分别监听 80 和 443 端口，服务器使用多线程支持多路并发同时处理多个服务请求。

### 2. 实验流程

(1) 根据上述要求，参照 http-server-example.c 编写 HTTP 服务器程序

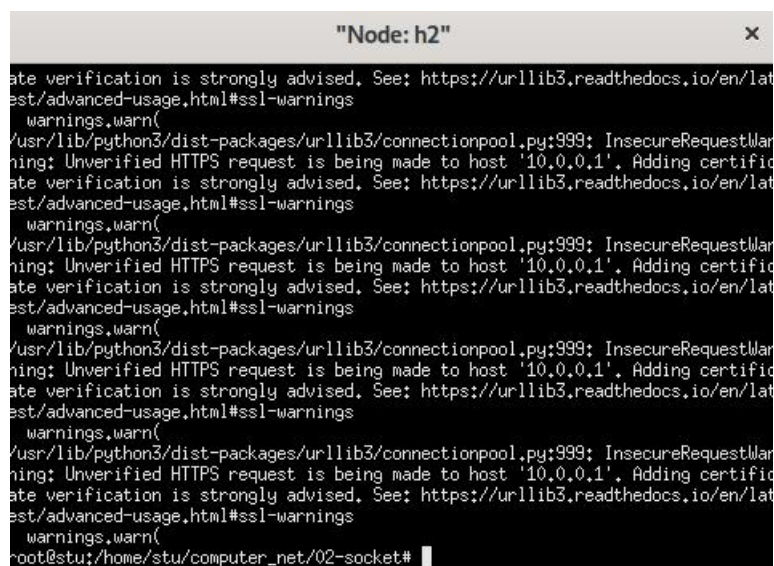
(2) 执行 `sudo python2 topo.py` 命令，

然后输入 `xtrem h1 h2` 生成包括两个端节点的网络拓扑，在主机 h1 上运行 HTTP 服务器程序，同时监听 80 和 443 端口

`h1 # ./http-server`

在主机 h2 上运行测试程序，验证 test.py 中的 7 个测试以验证程序正确性

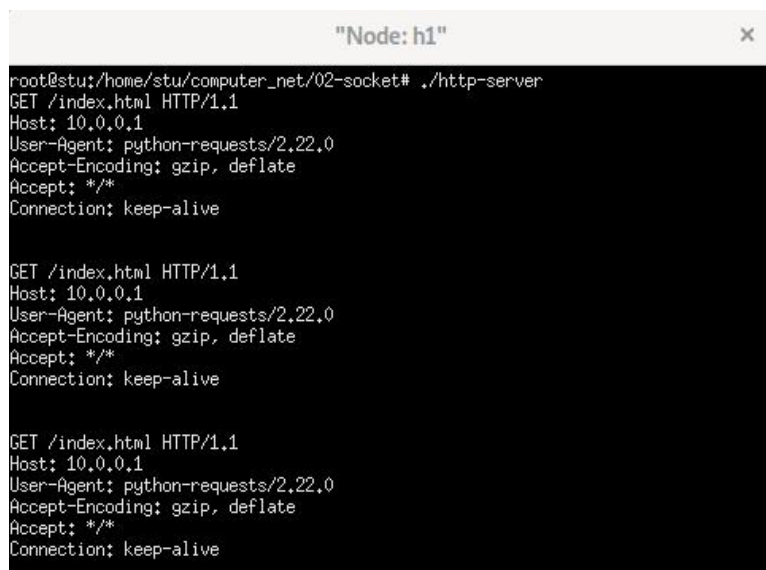
`h2 # python3 test/test.py`



```
"Node: h2"
ate verification is strongly advised. See: https://urllib3.readthedocs.io/en/lat
est/advanced-usage.html#ssl-warnings
warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWar
ning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certifi
ate verification is strongly advised. See: https://urllib3.readthedocs.io/en/lat
est/advanced-usage.html#ssl-warnings
warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWar
ning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certifi
ate verification is strongly advised. See: https://urllib3.readthedocs.io/en/lat
est/advanced-usage.html#ssl-warnings
warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWar
ning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certifi
ate verification is strongly advised. See: https://urllib3.readthedocs.io/en/lat
est/advanced-usage.html#ssl-warnings
warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWar
ning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certifi
ate verification is strongly advised. See: https://urllib3.readthedocs.io/en/lat
est/advanced-usage.html#ssl-warnings
warnings.warn(
root@stu:/home/stu/computer_net/02-socket#
```

图 1：主机 h2 中 test 程序正确执行

如果主机 h2 中没有出现 `AssertionError` 或其他错误, 则说明程序实现正确如图 1 所示, 主机 h1 作为监听接收端使用, 用作程序输出信息以便调试如图 2 所示打印接收到的 http 报文。



```

"Node: h1"
root@stu:/home/stu/computer_net/02-socket# ./http-server
GET /index.html HTTP/1.1
Host: 10.0.0.1
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

GET /index.html HTTP/1.1
Host: 10.0.0.1
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

GET /index.html HTTP/1.1
Host: 10.0.0.1
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

```

图 2: 主机 h1 输出得到的 http 报文

(3) 撰写实验报告

### 3. 设计实现

- ① 启动 https 的客户端, 同时开启 http 线程的客户端。https 服务器开启后需要先检验 ssl、证书和私钥, 检验无误之后再建立 socket 监听 443 端口, http 可以直接建立 socket 监听。
- ② HTTPS 服务器持续监听 443 端口, HTTP 持续监听 80 端口, 对于每个收到的服务请求验证无误后利用 `pthread()` 函数建立一个线程处理:

```

while (1)
{
    struct sockaddr_in caddr;
    socklen_t len;
    int csock = accept(sock, (struct sockaddr *)&caddr, &len);
    if (csock < 0)
    {
        perror("Accept failed");
        exit(1);
    }
    SSL *ssl = SSL_new(ctx);
    SSL_set_fd(ssl, csock);
    pthread_create(&https_pthread[(id++) % MAX_THREAD], NULL,
                  handle_https_request, (void *)ssl);
}

```

- ③ `handle_request` 应该支持处理 4 个不同的状态码如图 4 所示, 具体操作如下:
  - ① 解析 HTTP 请求报文得到 URL, 解析文件地址以及请求中是否有 Range 字段:

本次识别 URL 的方法十分简单，即按照 http 报文格式读到第一个 ‘/’ 为 URL 开始，读到 URL 后面的空格换行符为结束。识别 range: byte 字段可以采用 C 库的 <string.h> 文件中的 strstr 函数帮助识别字符串子串，再识别出 ‘= ‘即可获得需要读取的文件范围；

- ② 在本地利用 fopen() 函数寻找 URL 的路径下是否有对应的文件，若没有直接回复 404 数据包，若有则需要读取指定的文件到 send\_buffer，此操作可以利用 C 语言处理文件操作的函数解决，通过 fseek 移动 fp 指针得到文件大小，对于 range 只有读取起始字段的就读到文件结束，有读取结束字段的就利用得到的俩个字段计算需要读取的照度长度，然后将文件长度和 send\_buffer 的长度按照除余的方式分别写进 send\_buffer 即可。

```
int remainder = file_length % MAX_BUFFER;
int remain_flag = remainder != 0;
int result = file_length / MAX_BUFFER;

for (int i = 0; i < result; i++)
{
    fread(send_buffer, 1, MAX_BUFFER, fp);
    SSL_write(ssl, send_buffer, MAX_BUFFER);
}

if (remain_flag)
{
    fread(send_buffer, 1, remainder, fp);
    SSL_write(ssl, send_buffer, remainder);
}
```

- ③ 根据状态码和文件情况返回 HTTP 应答报文和数据：按照图 3 中的 HTTP 应答报文的格式以及图 4 中文件应该返回的状态码返回对应的 HTTP 应答报文以及刚刚读取到 send\_buffer 中的数据，发送方式参照样例即可，HTTP 报文的拼接需要注意 301 有 location 字段的加入。；

```
if (read_flag)
    strcat(send_buffer, "HTTP/1.1 206 Partial Content\r\nContent
-Length: ");
else
    strcat(send_buffer, "HTTP/1.1 200 OK\r\nContent-Length: ");
char file_len[10] = {0};
sprintf(&file_len, "%d", file_length);
strcat(send_buffer, file_len);
strcat(send_buffer, "\r\nConnection: keep-alive\r\nContent-Type:
text/html\r\n\r\n");
SSL_write(ssl, send_buffer, strlen(send_buffer));
```

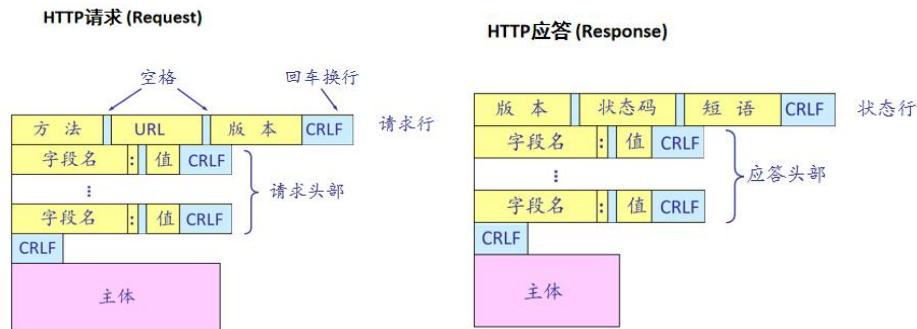


图 3: HTTP 请求和应答报文格式

需支持的状态码	场景
200 OK	对于443端口接收的请求，如果程序所在文件夹存在所请求的文件，返回该状态码，以及所请求的文件
301 Moved Permanently	对于80端口接收的请求，返回该状态码，在应答中使用 Location 字段表达相应的https URL
206 Partial Content	对于443端口接收的请求，如果所请求的为部分内容（请求中有Range字段），返回该状态码，以及相应的部分内容
404 Not Found	对于443端口接收的请求，如果程序所在文件夹没有所请求的文件，返回该状态码

图 4: 服务器支持的状态码

- ④ 采用 `pthread_create()`。一个主线程侦听一个端口号并接收请求，若干个子线程处理请求，因为 https 的服务器和 http 的服务器对 socket 的处理不同，https 需要用 SSL 加密，因此各自开启一个线程监听。
- ⑤ 程序结束阻塞线程 `pthread_join()`，利用 `close()` 函数关闭建立的 socket 描述符。

## 4. 分析讨论

本次识别 URL 然后再本地直接利用 URL 查找文件的方式是只针对此次的 test 文件设计的，是十分粗糙且不通用的，URL (Uniform Resource Locator) 是统一资源定位符的简称，有时候也被俗称为网页地址（网址），如同是网络上的门牌，是因特网上标准的资源的地址，通用的格式：`scheme://host[:port#]/path/.../[?query-string][#anchor]`，每个字段具体的含义在图 6 中说明。此次我直接识别 ‘/’ 后直接使用之后的内容作为查询位置，实际上这在大多数情况是不可行的，如网址 [www.mywebsite.com/sj/test/tes...](http://www.mywebsite.com/sj/test/tes...) 的各个字段在图 6 中说明，它所需的文件路径字段是需要多重解析的，因此如果我们需要使用 URL 中更具体的包括路径在内的其他字段的时候最好做出更好更具体的解析。

名称	对应的字段
Schema	http
host	www.mywebsite.com
path	/js/test/test.aspx
Query-string	name=sviergn&x=true
anchor	stuff

图 5: www.mywebsite.com/sj/test/tes…各个字段

名称	功能
scheme	访问服务器以获取资源时要使用哪种协议, 比如, http, https 和 FTP 等
host	HTTP 服务器的 IP 地址或域名
port#	HTTP 服务器的默认端口是 80, 这种情况下端口号可以省略, 如果使用了别的端口, 必须指明, 例如 <a href="http://www.cnblogs.com:8080">www.cnblogs.com: 8080</a>
path	访问资源的路径
query-string	发给 http 服务器的数据
anchor	锚

图 6: URL 通用格式

除此之外, HTTP 和 HTTPS 的区别也值得一提, 首先 HTTP 是互联网上应用最为广泛的一种网络协议, 是一个客户端和服务端请求和应答的标准 (TCP), 用于从 WWW 服务器传输超文本到本地浏览器的传输协议, 它可以使浏览器更加高效, 使网络传输减少。而 HTTPS 是以安全为目标的 HTTP 通道, 简单讲是 HTTP 的安全版, 即 HTTP 下加入 SSL 层, HTTPS 的安全基础是 SSL, 因此加密的详细内容就需要 SSL, HTTPS 协议的主要作用可以分为两种: 一种是建立一个信息安全通道, 来保证数据传输的安全; 另一种就是确认网站的真实性。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议, 要比 http 协议安全。本次实验中实例代码给出了简单的 SSL 处理实例。

显然, HTTPS 相比 HTTP 最大的不同就是多了一层 SSL (Secure Sockets Layer 安全套接层) 或 TLS (Transport Layer Security 安全传输层协议)。TLS 协议的建立需要经历 4 次握手。握手第一步是客户端向服务端发送 Client Hello 消息, 这个消息里包含



了一个客户端生成的随机数 Random1、客户端支持的加密套件和 SSL Version 等信息。通过 Wireshark 抓包，我们可以看到如下信息：

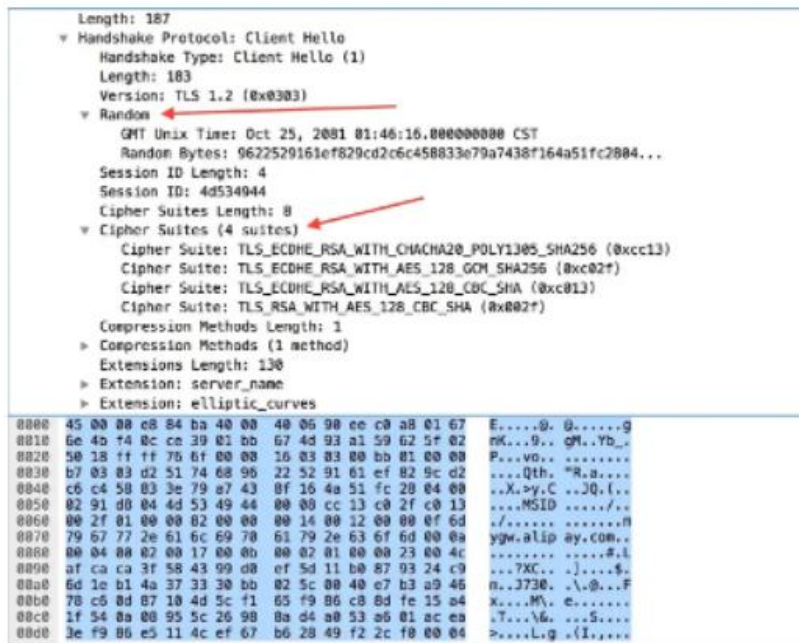


图 6：TLS 随机数 1

第二步是服务端向客户端发送 Server Hello 消息，这个消息会从 Client Hello 传过来的 Support Ciphers 里确定一份加密套件，这个套件决定了后续加密和生成摘要时具体使用哪些算法，另外还会生成一份随机数 Random2。注意，至此客户端和服务端都拥有了两个随机数（Random1+ Random2），这两个随机数会在后续生成对称密钥时用到。服务端将自己的证书下发给客户端，让客户验证自己的身份，客户端验证通过后取出证书中的公钥。

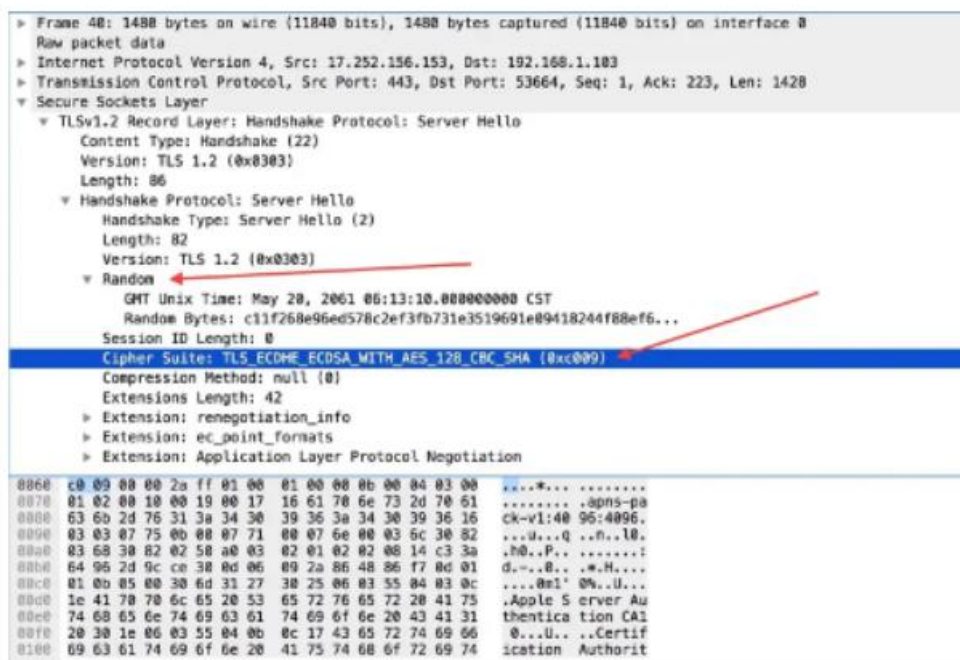


图 7：TLS 随机数 2

客户端收到服务端传来的证书后，先从 CA 验证该证书的合法性，验证通过后取出证书中的服务端公钥，再生成一个随机数  $Random_3$ ，再用服务端公钥非对称加密  $Random_3$  生成 **PreMaster Key**，客户端根据服务器传来的公钥生成了 **PreMaster Key**，**Client Key Exchange** 就是将这个 **key** 传给服务端，服务端再用自己的私钥解出这个 **PreMaster Key** 得到客户端生成的  $Random_3$ 。至此，客户端和服务端都拥有  $Random_1 + Random_2 + Random_3$ ，两边再根据同样的算法就可以生成一份密钥，握手结束后的应用层数据都是使用这个密钥进行对称加密。客户端通知服务端后面再发送的消息都会使用前面协商出来的密钥加密。到这里，双方已安全地协商出了同一份密钥，所有的应用层数据都会用这个密钥加密后再通过 TCP 进行可靠传输。

客户端发送 **Client Finish** 消息，客户端将前面的握手消息生成摘要再用协商好的密钥加密，服务端接收后会用密钥解密。服务端发送 **Server Finish** 消息，服务端也会将握手过程的消息生成摘要再用密钥加密，这是服务端发出的第一条加密消息。客户端接收后会用密钥解密，能解出来说明协商的密钥是一致的。  
过程简图可由图 8 所示。



图 8: TLS 建立链接 4 次握手

**参考资料：**

【1】 超详细的Socket通信原理和实例讲解：<https://zhuanlan.zhihu.com/p/100151937>。

【2】 struct sockaddr\_in结构体解析：[https://blog.csdn.net/u014748120/article/details/79447123?spm=1001.2101.3001.6661.1&utm\\_medium=distribute.pc\\_relevant\\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc\\_relevant\\_aa&depth\\_1-utm\\_source=distribute.pc\\_relevant\\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc\\_relevant\\_aa&utm\\_relevant\\_index=1](https://blog.csdn.net/u014748120/article/details/79447123?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_aa&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_aa&utm_relevant_index=1)。

【3】 pthread详解：<https://blog.csdn.net/networkhunter/article/details/100218945>。

【4】 <string.h>头文件详解  
<https://www.runoob.com/cprogramming/c-standard-library-string-h.html>。

【5】 C 语言文件操作函数 api 说明：  
<https://www.cnblogs.com/Kroner/p/10803695.html>。

【6】 HTTP 的一些解析：<https://juejin.cn/post/6844903511633707021>。

【7】 掘金社区 TLS 握手过程：<https://juejin.cn/post/6844904046063517704>。