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

# 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。

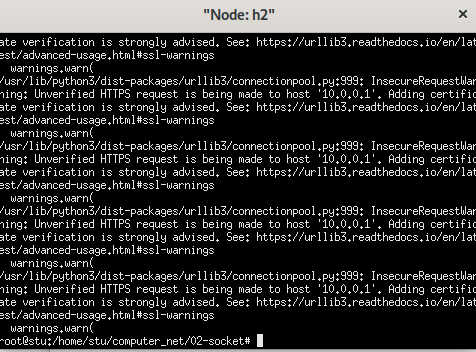
③服务器创建两个线程分别监听80和443端口，服务器使用多线程支持多路并发同时处理多个服务请求。

# 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，

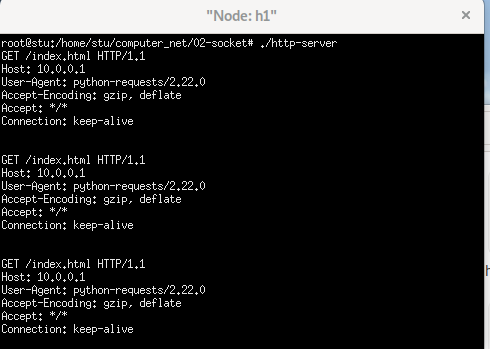
然后输入xtrem h1 h2生成包括两个端节点的网络拓扑，在主机h1上运行HTTP服务器程序，同时监听80和443端口

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



**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。



**图2: 主机h1输出得到的http报文**

（3）撰写实验报告

# 设计实现

1. 启动https的客户端，同时开启http线程的客户端。htttps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);

    }

1. 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字串，再识别出‘=‘即可获得需要读取的文件范围；

* 1. 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);

        }

* 1. 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: ");

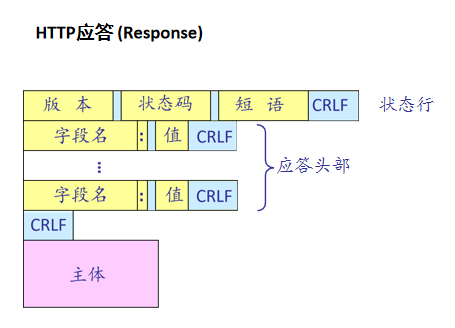
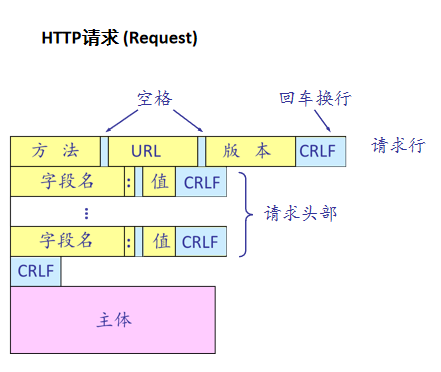
        char file\_len[10] = {0};

        sprintf(&file\_len, "%d", file\_length);

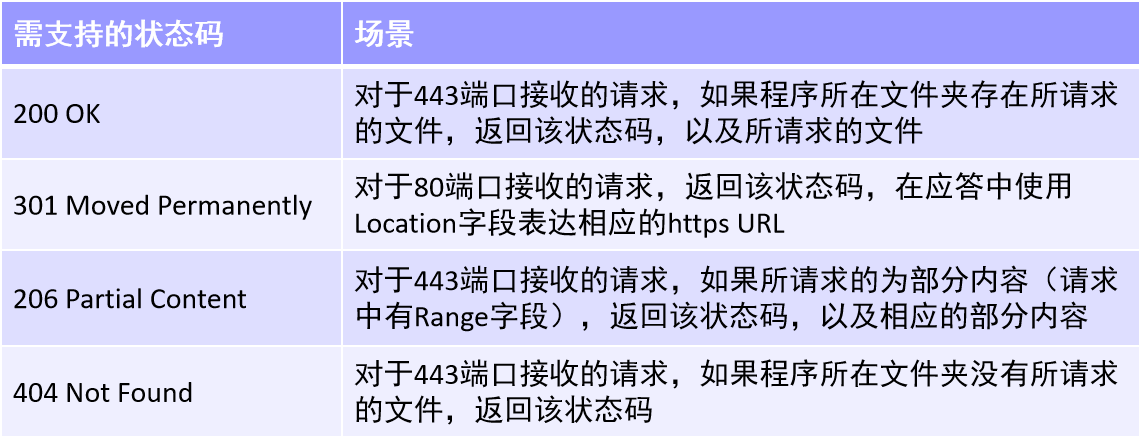
        strcat(send\_buffer, file\_len);

        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**



**图4：服务器支持的状态码**

1. 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。
2. 程序结束阻塞线程pthread\_join()，利用close()函数关闭建立的soket描述符，。

# 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…的各个字段在图6](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。



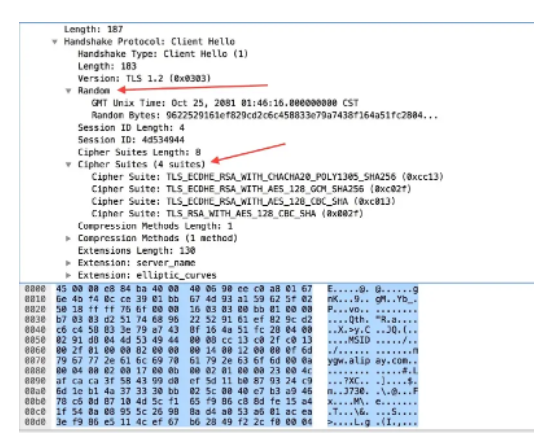
**图5：www.mywebsite.com/sj/test/tes…各个字段**



**图6：URL通用格式**

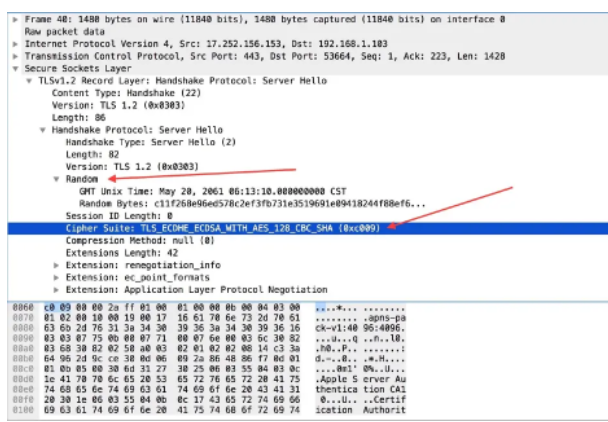
除此之外，HTTP和HTTPS的区别也值得一提，首先HTTP是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。而HTTPS是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL，HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http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秘钥时用到。服务端将自己的证书下发给客户端，让客户端验证自己的身份，客户端验证通过后取出证书中的公钥。

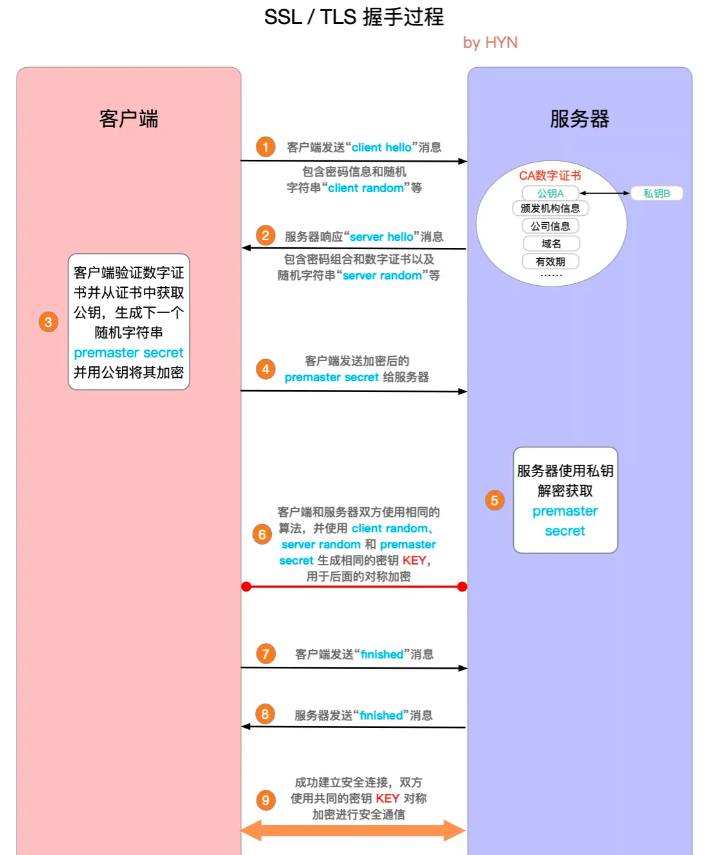


**图7：TLS随机数2**

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 Random3，再用服务端公钥非对称加密 Random3 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 Random3。至此，客户端和服务端都拥有 Random1 + Random2 + Random3，两边再根据同样的算法就可以生成一份秘钥，握手结束后的应用层数据都是使用这个秘钥进行对称加密。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秘钥加密。到这里，双方已安全地协商出了同一份秘钥，所有的应用层数据都会用这个秘钥加密后再通过 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秘钥加密，服务端接收后会用秘钥解密。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秘钥加密，这是服务端发出的第一条加密消息。客户端接收后会用秘钥解密，能解出来说明协商的秘钥是一致的。

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>。

【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>。