哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	HTTP 代理服务器的设计与实现					
姓名			院系	计算机科学与技术		
班级			学号			
任课教师			指导教师			
实验地点			实验时间			
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)					
教师评语						

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

熟悉并掌握 Socket 网络编程的过程与技术；深入理解 HTTP 协议，
掌握 HTTP 代理服务器的基本工作原理；掌握 HTTP 代理服务器设计与
编程实现的基本技能。

实验内容：

(1) 设计并实现一个基本 HTTP 代理服务器。 要求在指定端口（例如8080） 接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址所指向的 HTTP 服务器（原服务器）， 接收 HTTP 服务器的响应报文，并将响应报文转发给对应的客户进行浏览。

(2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since头行），向原服务器确认缓存对象是否是最新版本。

(3) 扩展 HTTP 代理服务器，支持如下功能：

a) 网站过滤：允许/不允许访问某些网站；

b) 用户过滤：支持/不支持某些用户访问外部网站；

c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）

实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

一．Socket 编程的客户端和服务器端主要步骤

客户端：

1. 根据目的服务器IP地址和端口号创建套接字，连接服务器
2. 发送请求报文
3. 接受返回报文
4. 关闭连接

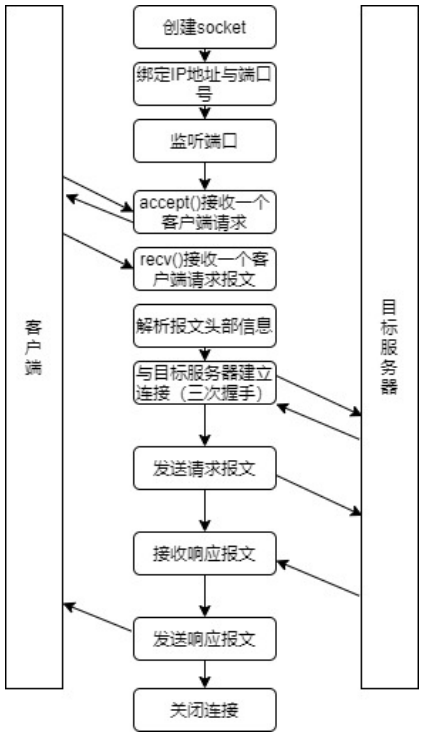
服务器端：

1. 创建套接字，绑定IP地址和端口号，监听端口
2. 从连接队列中取出一个连接请求，三次握手创建连接
3. 接收请求报文
4. 发送响应报文
5. 关闭当前连接，继续监听端口

二．HTTP 代理服务器的基本原理

代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接代理服务器在指定端口（例如 8080） 监听浏览器的访问请求（需要在客户端浏览器进行相应的设置）， 接收到浏览器对远程网站的浏览请求时，代理服务器开始在代理服务器的缓存中检索URL对应的对象（网页、图像等对象），找到对象文件后，提取该对象文件的最新被修改时间；代理服务器程序在客户的请求报文首部插入<If-Modified-Since: 对象文件的最新被修改时间>，并向原 Web 服务器转发修改后的请求报文。 如果代理服务器没有该对象的缓存，则会直接向原服务器转发请求报文，并将原服务器返回的响应直接转发给客户端，同时将对象缓存到代理服务器中。代理服务器程序会根据缓存的时间、大小和提取记录等对缓存进行清理。

三．HTTP 代理服务器的程序流程图



四．实现 HTTP 代理服务器的关键技术及解决方案

4.1 关键技术：代理服务器基本功能实现

解决方案：这一部分主要实现的功能就是上面第三部分中流程图中的全部内容，主要使用到的函数简单介绍如下：

Socket():根据选用的服务建立套接字，本实验使用的网络层协议是TCP

Bind():将一个本地地址与一个创建好的套接字绑定，使用这个函数意味着代理服务器端的套接字已经可以开始准备开始工作了

Listen():监听端口的连接请求，在使用这个函数之后套接字进入监听模式

Accept():在一个套接字处接受连接请求，注意和listen的区别，在accept()函数起作用之后意味着代理服务器和客户端之间的连接已经建立起来了。

Connect()：建立socket连线

Send():发送报文（本实验中使用的时候主要是向目标服务器发送请求报文，向客户端发送响应报文）

Recv():接收报文

接下来简单介绍代理服务器的工作流程：

1. 首先初始化套接字。使用socket函数创建一个套接字，并利用bind函数将套接字和代理服务器的本地地址绑定，在本实验中根据实验指导书，将代理服务器的IP地址设置为“127.0.0.1”，将端口设置为10240。在这一步完成之后套接字初始化完成，使用listen函数，套接字进入监听模式，等待客户端发送的连接请求。
2. 进入监听模式之后使用accept函数对于每个来到的请求进行接收，在本实验中为了保证效率，所以对于每一个连接都创建一个单独的进程来提供服务
3. 当连接建立之后，使用recv函数接收客户端发送的HTTP报文，解析出其中的目的服务器的相关信息，并使用connect函数与目的服务器建立连接，使用send函数将HTTP请求报文发送给目的服务器。
4. 等待目的服务器返回响应报文，使用send函数将响应报文发送给客户端。如果前述过程一切正常，表示代理服务器的任务已经完成。

5. 处理完成之后，等待200ms之后，关闭该线程，清理缓存，可以继续处理接下来的连接请求。

4.2 关键技术：实现代理服务器缓存

解决方案：代理服务器缓存技术在选做内容中相对是比较复杂的，实现过程主要如下：

1. 创建一个cache数组用来存储用户访问过的服务器中的数据
2. 当客户第一次访问某个服务器中的数据的时候，代理服务器将返回报文中的内容缓存下来，保存到本地。在本实验中由于是直接在程序中存储，因此设置的存储容量比较小。
3. 如果在cache数组中没有查找到代理服务器已经缓存过服务器的内容，和2同操作；如果查询到已经有缓存，需要解决这个缓存是否已经不是目的服务器上最新的内容了。因此本实验中解析客户端发送的HTTP请求报文，加入一个“If-Modified-Since”，并加上缓存中的最后修改时间，以此询问目的服务器是否在这段时间内发生了更改，如果目的服务器返回状态码304，说明代理服务器缓存的就是最新的，那么就将缓存返回给客户端；如果返回状态码200，表示已经更新，那么将更新过后的响应内容发送给客户端，并更新缓存。

4.3 关键技术：网站过滤

解决方案：在代理服务器段对于客户发送的请求报文进行解析，提取出其要访问的url，如果和已知需要过滤的网站一致，直接关闭套接字，结束这次连接，代码片段如下：

```
if (strstr(httpHeader->url, invalid_website[0]) != NULL)
    {
        printf("\n====================\n");
        printf("--------该网站已被屏蔽!----------\n");
        goto error;
    }
```

4.4 关键技术：用户过滤

解决方案：本实验实现过程中是实现了只能从限定的部分用户才能访问，因此只需要在建立连接的时候比对请求连接的客户端IP地址是否和限定的IP地址一致，如果一致允许继续执行程序；如果不一致直接断开连接。代码片段如下：

```
if (!strcmp(restrict_host[0], inet_ntoa(addr_in.sin_addr)) && button)
        {
            printf("该用户访问受限\n");
            continue;
        }
```

4.5 关键技术：网站引导

解决方案：检测客户端发送的HTTP请求报文，如果发现访问的网址是要被钓鱼的网址，则将该网址引导到其他网站（钓鱼目的网址），通过更改 HTTP 头部字段的 url和主机名来实现，部分代码如下：

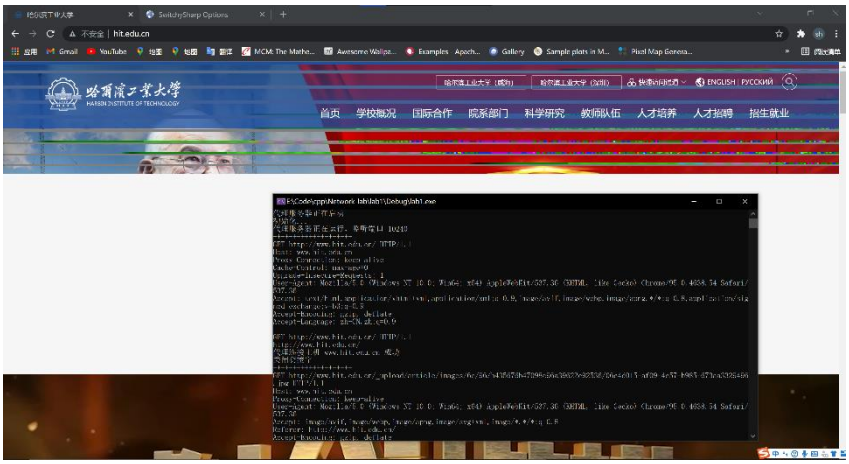
```
if (strstr(httpHeader->url, fishing_src) != NULL)
    {
        printf("\n====================\n");
        printf("-------------已从源网址：%s 转到 目的网址 ：%s ----------------
    \n", fishing_src, fishing_dest);
        memcpy(httpHeader->host, fishing_dest_host, strlen(fishing_dest_host) + 1);
        memcpy(httpHeader->url, fishing_dest, strlen(fishing_dest));
```

```
7.      }
```

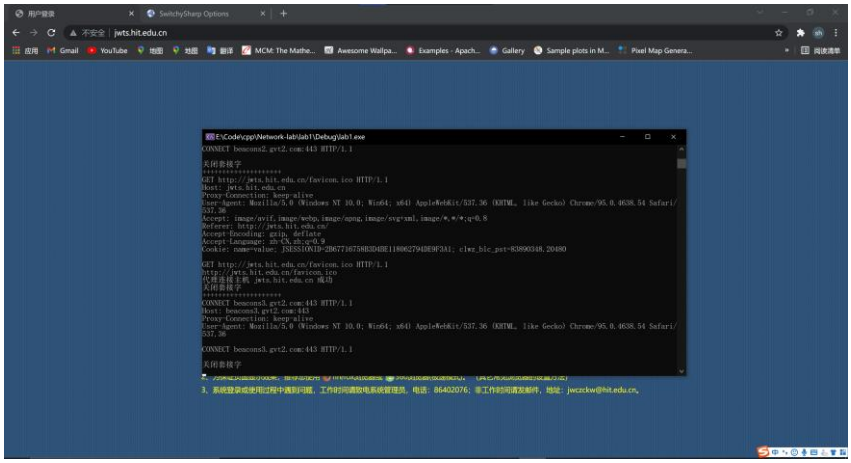
实验结果：

（1）首先在实验开始之前设置浏览器连接代理服务器的IP地址和端口，如下图所示：

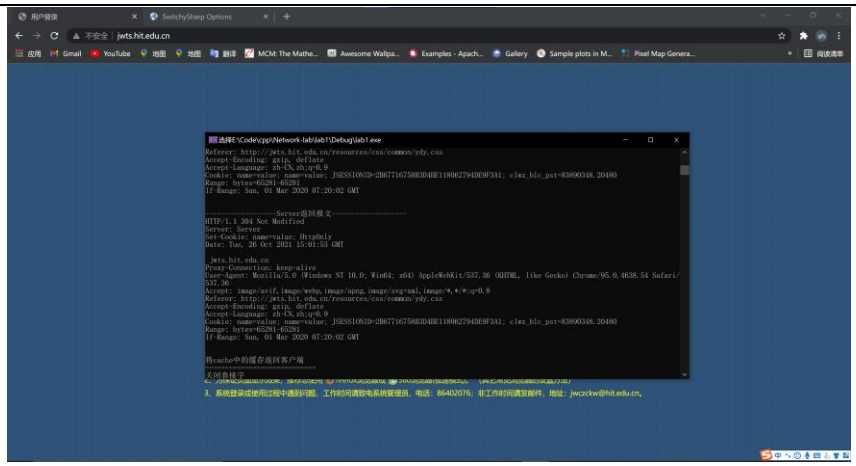
（2）接着是HTTP代理服务器基本功能实现，连接的网址为：http://www.hit.edu.cn，显示结果为：



（3）接着测试支持cache的HTTP代理服务器的功能，访问网址为：http://jwts.hit.edu.cn，第一次访问结果：



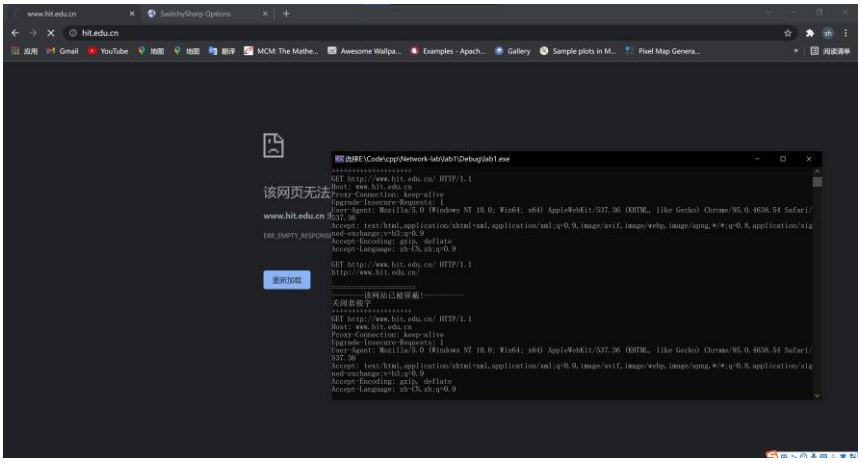
第二次访问结果如下：



观察目的服务器段返回的报文可以知道，访问的网站中内容和缓存的内容是一致的，因此直接将缓存的内容返回给客户端。

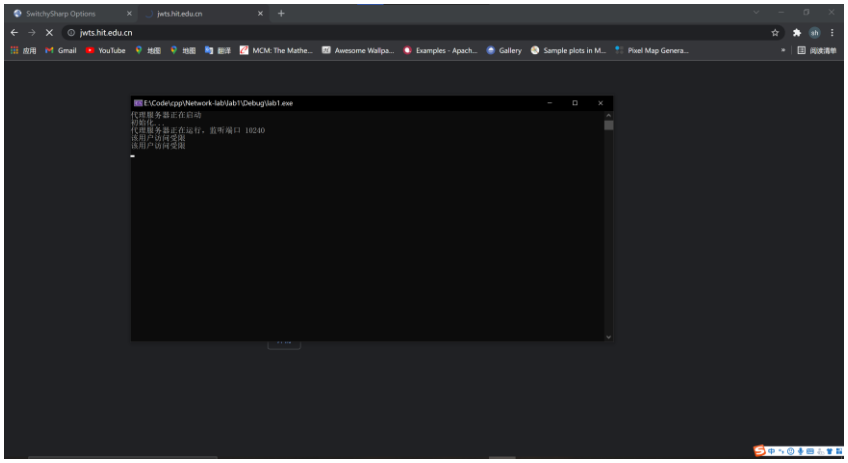
（4）扩展HTTP代理服务器：

（a）网站过滤，不允许访问http://www.hit.edu.cn，可以知道我们上述测试中这网站是可以正常访问的，而当开启了网站过滤功能之后结果如下：

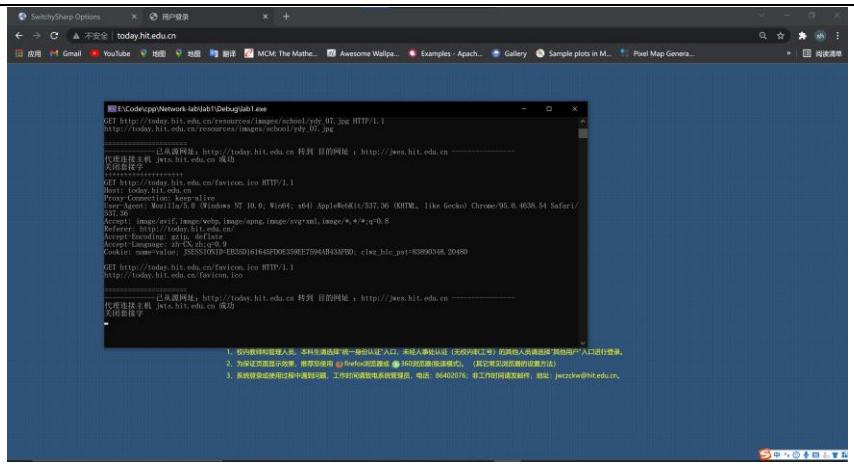


可以发现网站已经无法访问，而且在控制台中输出了我们预期的网站被屏蔽的提示信息。

（b）用户过滤，将本机地址屏蔽，访问地址为http://jwts.hit.edu.cn/，经上述测试，如果不使用屏蔽的时候访问结果正常，而当开启了用户过滤之后可以发现结果如下，已经无法访问，且在控制台输出相应的提示信息：



（c）网站引导：将网址 http://today.hit.edu.cn/ 的访问引导到网址 http://jwts.hit.edu.cn/ ，运行结果如图：



可以观察到网站被引导到了http://jwts.hit.edu.cn/，在控制台也输出了相关的提示信息，证明引导成功。

问题讨论：

对实验过程中的思考问题进行讨论或回答。

（1）对于指导书中的参考代码报错的一些思考：a）报错的第一个点就是#include "stdafx.h"这一行代码报错，解决方式就是直接去掉即可。错误的原因是没有这个头文件，而这个头文件的作用经查询资料可知，Windows和MFC的include文件都非常大，即使有一个快速的处理程序，编译程序也要花费相当长的时间来完成工作。由于每个.CPP文件都包含相同的include文件，为每个.CPP文件都重复处理这些文件耗时很长，因此stdafx.h可以视为一个包含所有需要include的头文件，在编译开始的时候先把这些内容编译好，之后在编译每一个cpp文件的时候就阅读编译好的结果即可。b）报错的第二个比较大的点就是代码中使用了goto，而且在调用goto语句之后还定义了随机变量，大部分编译器都会认为这种行为是危险的，而在VS2019中可以通过对其进行简单设置去除这种错误，经查阅资料，其余一些编译器需要将所有的随机变量都定义在调用goto之前（事实上这样才是真正安全的，VS2019的处理方式只是强制略过这种错误检测）。

（2）对于使用代理服务器访问一些网站的时候存在图片无法加载或者加载的很慢的情况，猜测是因为对于代理服务器来说，HTTP发送报文的时候是不采用流水的，因此中间需要多次发送和接受过程，而且还经过了代理服务器这一中介，可能使得速度更慢，因此在现在的HTTP报文传送中猜测大多使用了流水线形式或其他形式加速发送报文。

心得体会：

结合实验过程和结果给出实验的体会和收获。

本次实验让我对于socket编程的基本函数和基本操作有了认识，同时也了解了HTTP代理服务器的基本原理，对于HTTP请求和响应过程有了更深刻的认识。同时通过一些杜宇代理服务器的扩展，对于网站引导和屏蔽的最简单的实现方式有了认识，对于socket编程产生了浓厚兴趣。

代码实现如下：

```
1.  #include <stdio.h>
2.  #include <Windows.h>
3.  #include <process.h>
4.  #include <string.h>
5.  #pragma comment(lib,"Ws2_32.lib")
6.  #define MAXSIZE 65507 //发送数据报文的最大长度
7.  #define HTTP_PORT 80 //http 服务器端口
8.  #define DATELENGTH 50 //时间字节数
9.  #define CACHE_NUM 50  //定义最大缓存数量
```

```
10. //Http 重要头部数据
11. struct HttpHeader {
12.     char method[4]; // POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
13.     char url[1024]; // 请求的 url
14.     char host[1024]; // 目标主机
15.     char cookie[1024 * 10]; //cookie
16.     HttpHeader() {
17.         ZeroMemory(this, sizeof(HttpHeader));
18.     }
19. };
20. //因为不做外部存储，所以为了节省空间，cache 存储的时候
21. //去掉 Http 头部信息中的 cookie
22. struct cacheHttpHead {
23.     char method[4]; // POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
24.     char url[1024]; // 请求的 url
25.     char host[1024]; // 目标主机
26.     cacheHttpHead() {
27.         ZeroMemory(this, sizeof(cacheHttpHead));
28.     }
29. };
30. //代理服务器缓存技术
31. struct CACHE {
32.     cacheHttpHead httpHead;
33.     char buffer[MAXSIZE];//储存报文返回内容
34.     char date[DATELENGTH];//缓存内容的最后修改时间
35.     CACHE() {
36.         ZeroMemory(this->buffer, MAXSIZE);
37.         ZeroMemory(this->date, DATELENGTH);
38.     }
39. };
40. CACHE cache[CACHE_NUM];//缓存地址
41. int cache_index = 0;//记录当前应该将缓存放在哪个位置
42.
43. BOOL InitSocket();
44. void ParseHttpHead(char* buffer, HttpHeader* httpHeader);
45. BOOL ConnectToServer(SOCKET* serverSocket, char* host);
46. unsigned int __stdcall ProxyThread(LPVOID lpParameter);
47. int isInCache(CACHE* cache, HttpHeader httpHeader);//寻找缓存中是否存在，如果存在返回 index，不存在返回-1
48. BOOL httpEqual(cacheHttpHead http1, HttpHeader http2);//判断两个 http 报文是否相同,主要用来判断缓存和报文是否相同
49. void changeHTTP(char* buffer, char* date);//用于修改 HTTP 报文
50. //代理相关参数
51. SOCKET ProxyServer;
```

```
52. sockaddr_in ProxyServerAddr;
53. const int ProxyPort = 10240;
54. //由于新的连接都使用新线程进行处理，对线程的频繁的创建和销毁特别浪费资源
55. //可以使用线程池技术提高服务器效率
56. //const int ProxyThreadMaxNum = 20;
57. //HANDLE ProxyThreadHandle[ProxyThreadMaxNum] = {0};
58. //DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};
59. struct ProxyParam {
60.     SOCKET clientSocket;
61.     SOCKET serverSocket;
62. };
63. //选做功能参数定义
64. bool button =true;//取 true 的时候表示开始运行选做功能
65. //禁止访问网站
66. char* invalid_website[10] = { "http://www.hit.edu.cn" };
67. const int invalid_website_num = 1;//有多少个禁止网站
68. //钓鱼网站
69. char* fishing_src = "http://today.hit.edu.cn";//钓鱼网站原网址
70. char* fishing_dest = "http://jwes.hit.edu.cn";//钓鱼网站目标网址
71. char* fishing_dest_host = "jwts.hit.edu.cn";//钓鱼目的地址主机名
72. //限制访问用户
73. char* restrict_host[10] = { "127.0.0.1" };
74. int main(int argc, char* argv[])
75. {
76.     printf("代理服务器正在启动\n");
77.     printf("初始化...\n");
78.     if (!InitSocket()) {
79.         printf("socket 初始化失败\n");
80.         return -1;
81.     }
82.     printf("代理服务器正在运行，监听端口 %d\n", ProxyPort);
83.     SOCKET acceptSocket = INVALID_SOCKET;
84.     ProxyParam* lpProxyParam;
85.     HANDLE hThread;
86.     DWORD dwThreadID;
87.     sockaddr_in addr_in;
88.     int addr_len = sizeof(SOCKADDR);
89.     //代理服务器不断监听
90.     while (true) {
91.         acceptSocket = accept(ProxyServer, (SOCKADDR*)&addr_in, &(addr_len))
    ;
92.         lpProxyParam = new ProxyParam;
93.         if (lpProxyParam == NULL) {
94.             continue;
```

```
95.         }
96.         //受限用户,与列表中匹配上的都无法访问
97.         if (strcmp(restrict_host[0], inet_ntoa(addr_in.sin_addr)) && button)
    //注意比较之前将网络二进制的数字转换成网络地址
98.         {
99.             printf("该用户访问受限\n");
100.             continue;
101.         }
102.         lpProxyParam->clientSocket = acceptSocket;
103.         hThread = (HANDLE)_beginthreadex(NULL, 0,
104.             &ProxyThread, (LPVOID)lpProxyParam, 0, 0);
105.         CloseHandle(hThread);
106.         Sleep(200);
107.     }
108.     closesocket(ProxyServer);
109.     WSACleanup();
110.     return 0;
111. }
112. //************************************
113. // Method: InitSocket
114. // FullName: InitSocket
115. // Access: public
116. // Returns: BOOL
117. // Qualifier: 初始化套接字
118. //************************************
119. BOOL InitSocket() {
120.     //加载套接字库（必须）
121.     WORD wVersionRequested;
122.     WSADATA wsaData;
123.     //套接字加载时错误提示
124.     int err;
125.     //版本 2.2
126.     wVersionRequested = MAKEWORD(2, 2);
127.     //加载 dll 文件 Scoket 库
128.     err = WSAStartup(wVersionRequested, &wsaData);
129.     if (err != 0) {
130.         //找不到 winsock.dll
131.         printf("加载 winsock 失败，错误代码为: %d\n", WSAGetLastError());
132.         return FALSE;
133.     }
134.     //if 中的语句主要用于比对是否是 2.2 版本
135.     if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
136.     {
137.         printf("不能找到正确的 winsock 版本\n");
```

```
138.         WSACleanup();
139.         return FALSE;
140.     }
141.     //创建的 socket 文件描述符基于 IPV4，TCP
142.     ProxyServer = socket(AF_INET, SOCK_STREAM, 0);
143.     if (INVALID_SOCKET == ProxyServer) {
144.         printf("创建套接字失败，错误代码为： %d\n", WSAGetLastError());
145.         return FALSE;
146.     }
147.     ProxyServerAddr.sin_family = AF_INET;
148.     ProxyServerAddr.sin_port = htons(ProxyPort);//整型变量从主机字节顺序转变成网络字节顺序,转换为大端法
149.     ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;//泛指本机也就是表示本机的所有 IP，多网卡的情况下，这个就表示所有网卡 ip 地址的意思
150.     if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
151.         printf("绑定套接字失败\n");
152.         return FALSE;
153.     }
154.     if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR) {
155.         printf("监听端口%d 失败", ProxyPort);
156.         return FALSE;
157.     }
158.     return TRUE;
159. }
160. //************************************
161. // Method: ProxyThread
162. // FullName: ProxyThread
163. // Access: public
164. // Returns: unsigned int __stdcall
165. // Qualifier: 线程执行函数
166. // Parameter: LPVOID lpParameter
167. //************************************
168. unsigned int __stdcall ProxyThread(LPVOID lpParameter) {
169.     char Buffer[MAXSIZE];
170.     char* CacheBuffer;
171.     ZeroMemory(Buffer, MAXSIZE);
172.     SOCKADDR_IN clientAddr;
173.     int length = sizeof(SOCKADDR_IN);
174.     int recvSize;
175.     int ret;
176.     recvSize = recv(((ProxyParam
177.         *)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);//接收到报文
178.     if (recvSize <= 0) {
```

```
179.         goto error;
180.     }
181.     HttpHeader* httpHeader = new HttpHeader();
182.     CacheBuffer = new char[recvSize + 1];
183.     ZeroMemory(CacheBuffer, recvSize + 1);
184.     memcpy(CacheBuffer, Buffer, recvSize);
185.     //处理 HTTP 头部
186.     ParseHttpHead(CacheBuffer, httpHeader);
187.     //处理禁止访问网站
188.     if (strstr(httpHeader->url, invalid_website[0]) != NULL && button)
189.     {
190.         printf("\n====================\n");
191.         printf("--------该网站已被屏蔽!----------\n");
192.         goto error;
193.     }
194.     //处理钓鱼网站
195.     if (strstr(httpHeader->url, fishing_src) != NULL && button)
196.     {
197.         printf("\n====================\n");
198.         printf("-------------已从源网址：%s 转到 目的网址 ： %s --------------
    --\n", fishing_src, fishing_dest);
199.         //修改 HTTP 报文
200.         memcpy(httpHeader->host, fishing_dest_host, strlen(fishing_dest_hos
    t) + 1);
201.         memcpy(httpHeader->url, fishing_dest, strlen(fishing_dest));
202.     }
203.     delete CacheBuffer;
204.     //连接目标主机
205.     if (!ConnectToServer(&((ProxyParam
206.         *)lpParameter)->serverSocket, httpHeader->host)) {
207.         goto error;
208.     }
209.     printf("代理连接主机 %s 成功\n", httpHeader->host);
210.
211.     int index = isInCache(cache, *httpHeader);
212.     //如果在缓存中存在
213.     if (index > -1)
214.     {
215.         char* cacheBuffer;
216.         char Buf[MAXSIZE];
217.         ZeroMemory(Buf, MAXSIZE);
218.         memcpy(Buf, Buffer, recvSize);
219.         //插入"If-Modified-Since: "
220.         changeHTTP(Buf, cache[index].date);
```

```
221.          printf("-------------------请求报文-----------------------
    \n%s\n", Buf);
222.          ret = send(((ProxyParam
223.              *)lpParameter)->serverSocket, Buf, strlen(Buf)+1, 0);
224.          recvSize = recv(((ProxyParam
225.              *)lpParameter)->serverSocket, Buf, MAXSIZE, 0);
226.          printf("------------------Server 返回报文------------------
    \n%s\n", Buf);
227.          if (recvSize <= 0) {
228.              goto error;
229.          }
230.          char* No_Modified = "304";
231.          //没有改变，直接返回 cache 中的内容
232.          if (!memcmp(&Buf[9], No_Modified, strlen(No_Modified)))
233.          {
234.              ret = send(((ProxyParam
235.                  *)lpParameter)->clientSocket, cache[index].buffer, strlen(c
    ache[index].buffer) + 1, 0);
236.              printf("将 cache 中的缓存返回客户端\n");
237.              printf("============================\n");
238.              goto error;
239.          }
240.      }
241.      //将客户端发送的 HTTP 数据报文直接转发给目标服务器
242.      ret = send(((ProxyParam*)lpParameter)->serverSocket, Buffer, strlen(Buf
    fer)
243.          + 1, 0);
244.      //等待目标服务器返回数据
245.      recvSize = recv(((ProxyParam
246.          *)lpParameter)->serverSocket, Buffer, MAXSIZE, 0);
247.      if (recvSize <= 0) {
248.          goto error;
249.      }
250.      //以下部分将返回报文加入缓存
251.      //从服务器返回报文中解析时间
252.      char* cacheBuffer2 = new char[MAXSIZE];
253.      ZeroMemory(cacheBuffer2, MAXSIZE);
254.      memcpy(cacheBuffer2, Buffer, MAXSIZE);
255.      char* delim = "\r\n";
256.      char date[DATELENGTH];
257.      char* nextStr;
258.      ZeroMemory(date, DATELENGTH);
259.      char* p = strtok_s(cacheBuffer2, delim, &nextStr);
260.      bool flag = false;//表示是否含有修改时间报文
```

```
261.     //不断分行，直到分出具有修改时间的那一行
262.     while (p)
263.     {
264.         if (p[0] == 'L')//找到 Last-Modified:那一行
265.         {
266.             if (strlen(p) > 15)
267.             {
268.                 char header[15];
269.                 ZeroMemory(header, sizeof(header));
270.                 memcpy(header, p, 14);
271.                 if (!(strcmp(header, "Last-Modified:")))
272.                 {
273.                     memcpy(date, &p[15], strlen(p) - 15);
274.                     flag = true;
275.                     break;
276.                 }
277.             }
278.         }
279.         p = strtok_s(NULL, delim, &nextStr);
280.     }
281.     if (flag)
282.     {
283.         if (index > -1)//说明已经有内容存在，只要改一下时间和内容
284.         {
285.             memcpy(&(cache[index].buffer), Buffer, strlen(Buffer));
286.             memcpy(&(cache[index].date), date, strlen(date));
287.         }
288.         else//第一次访问，需要完全缓存
289.         {
290.             memcpy(&(cache[cache_index % CACHE_NUM].httpHead.host), httpHea
    der->host,strlen(httpHeader->host));
291.             memcpy(&(cache[cache_index % CACHE_NUM].httpHead.method), httpH
    eader->method, strlen(httpHeader->method));
292.             memcpy(&(cache[cache_index % CACHE_NUM].httpHead.url), httpHead
    er->url, strlen(httpHeader->url));
293.             memcpy(&(cache[cache_index % CACHE_NUM].buffer), Buffer, strlen
    (Buffer));
294.             memcpy(&(cache[cache_index % CACHE_NUM].date), date, strlen(dat
    e));
295.             cache_index++;
296.         }
297.     }
298.     //将目标服务器返回的数据直接转发给客户端
299.     ret = send(((ProxyParam
```

```
300.         *)lpParameter)->clientSocket, Buffer, sizeof(Buffer), 0);
301.     //错误处理
302. error:
303.     printf("关闭套接字\n");
304.     Sleep(200);
305.     closesocket(((ProxyParam*)lpParameter)->clientSocket);
306.     closesocket(((ProxyParam*)lpParameter)->serverSocket);
307.     delete lpParameter;
308.     _endthreadex(0);
309.     return 0;
310. }
311. //************************************
312. // Method: ParseHttpHead
313. // FullName: ParseHttpHead
314. // Access: public
315. // Returns: void
316. // Qualifier: 解析 TCP 报文中的 HTTP 头部
317. // Parameter: char * buffer
318. // Parameter: HttpHeader * httpHeader
319. //************************************
320. void ParseHttpHead(char* buffer, HttpHeader* httpHeader) {
321.     char* p;
322.     char* ptr;
323.     const char* delim = "\r\n";
324.     p = strtok_s(buffer, delim, &ptr);//提取第一行
325.     printf("%s\n", p);
326.     if (p[0] == 'G') {//GET 方式
327.         memcpy(httpHeader->method, "GET", 3);
328.         memcpy(httpHeader->url, &p[4], strlen(p) - 13);
329.     }
330.     else if (p[0] == 'P') {//POST 方式
331.         memcpy(httpHeader->method, "POST", 4);
332.         memcpy(httpHeader->url, &p[5], strlen(p) - 14);
333.     }
334.     printf("%s\n", httpHeader->url);
335.     p = strtok_s(NULL, delim, &ptr);
336.     while (p) {
337.         switch (p[0]) {
338.         case 'H'://Host
339.             memcpy(httpHeader->host, &p[6], strlen(p) - 6);
340.             break;
341.         case 'C'://Cookie
342.             if (strlen(p) > 8) {
343.                 char header[8];
```

```
344.                 ZeroMemory(header, sizeof(header));
345.                 memcpy(header, p, 6);
346.                 if (!strcmp(header, "Cookie")) {
347.                     memcpy(httpHeader->cookie, &p[8], strlen(p) - 8);
348.                 }
349.             }
350.             break;
351.         default:
352.             break;
353.         }
354.         p = strtok_s(NULL, delim, &ptr);
355.     }
356. }
357. //************************************
358. // Method: ConnectToServer
359. // FullName: ConnectToServer
360. // Access: public
361. // Returns: BOOL
362. // Qualifier: 根据主机创建目标服务器套接字，并连接
363. // Parameter: SOCKET * serverSocket
364. // Parameter: char * host
365. //************************************
366. BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
367.     sockaddr_in serverAddr;
368.     serverAddr.sin_family = AF_INET;
369.     serverAddr.sin_port = htons(HTTP_PORT);
370.     HOSTENT* hostent = gethostbyname(host);
371.     if (!hostent) {
372.         return FALSE;
373.     }
374.     in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
375.     serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));//将一个将网络
    地址转换成一个长整数型数
376.     *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
377.     if (*serverSocket == INVALID_SOCKET) {
378.         return FALSE;
379.     }
380.     if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr))

381.         == SOCKET_ERROR) {
382.         closesocket(*serverSocket);
383.         return FALSE;
384.     }
385.     return TRUE;
```

```
386. }
387. BOOL httpEqual(cacheHttpHead http1, HttpHeader http2)
388. {
389.     if (strcmp(http1.method, http2.method))return false;
390.     if (strcmp(http1.host, http2.host))return false;
391.     if (strcmp(http1.url, http2.url))return false;
392.     return true;
393. }
394. int isInCache(CACHE* cache, HttpHeader httpHeader)
395. {
396.     int index = 0;
397.     for (; index < CACHE_NUM; index++)
398.     {
399.         if (httpEqual(cache[index].httpHead, httpHeader))return index;
400.     }
401.     return -1;
402. }
403. void changeHTTP(char* buffer, char* date)
404. {
405.     //此函数在 HTTP 中间插入"If-Modified-Since: "
406.     const char* strHost = "Host";
407.     const char* inputStr = "If-Modified-Since: ";
408.     char temp[MAXSIZE];
409.     ZeroMemory(temp, MAXSIZE);
410.     char* pos = strstr(buffer, strHost);//找到 Host 位置
411.     int i = 0;
412.     //将 host 与之后的部分写入 temp
413.     for (i = 0; i < strlen(pos); i++) {
414.         temp[i] = pos[i];
415.     }
416.     *pos = '\0';
417.     while (*inputStr != '\0') {  //插入 If-Modified-Since 字段
418.         *pos++ = *inputStr++;
419.     }
420.     while (*date != '\0') {
421.         *pos++ = *date++;
422.     }
423.     *pos++ = '\r';
424.     *pos++ = '\n';
425.     //将 host 之后的字段复制到 buffer 中
426.     for (i = 0; i < strlen(temp); i++) {
427.         *pos++ = temp[i];
428.     }
429. }
```