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			
班级			学号			
任课教师			指导教师			
实验地点	格物 207		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；

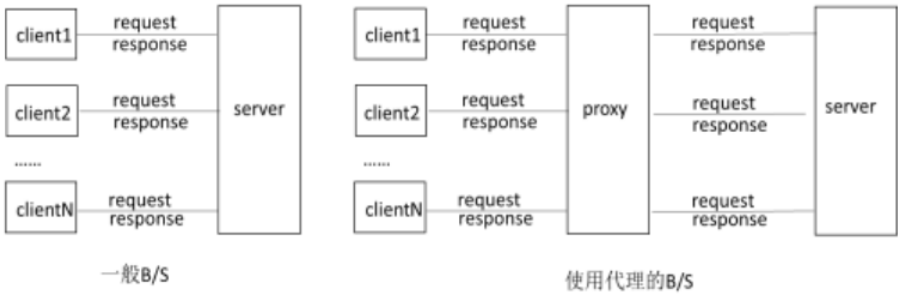
b) 用户过滤：支持/不支持某些用户访问外部网站；

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：

1. 基本功能实现：

代理服务器的工作方式如下：



a. InitSocket函数实现代理服务器创建HTTP代理服务的TCP主套接字，然后通过该主套接字等待客户端的连接请求：

创建套接字：

```
ProxyServer = socket(AF_INET, SOCK_STREAM, 0);

if (INVALID_SOCKET == ProxyServer)
{
    printf("创建套接字失败，错误代码为： %d\n", WSAGetLastError());
    return FALSE;
}
```

设置代理服务器的网络通信地址：

```
ProxyServerAddr.sin_family = AF_INET;
ProxyServerAddr.sin_port = htons(ProxyPort);     //将一个无符号短整型数值
//ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;  //INADDR_ANY == 0.0
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.3");  //仅限本
```

绑定套接字与网络地址并监听：

```
//绑定套接字与网络地址
if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
{
    printf("绑定套接字失败\n");
    return FALSE;
}

//监听套接字
if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR)
{
    printf("监听端口%d 失败", ProxyPort);
    return FALSE;
}
```

b. ParseHttpHead函数读取客户端的HTTP请求报文，通过请求行中的 URL， 解析客户期望访问的原服务器 IP 地址：

第一行读取请求和URL：

```
if (p[0] == 'G')
{ // GET 方式
    memcpy(httpHeader->method, "GET", 3);
    memcpy(httpHeader->url, &p[4], strlen(p) - 13);
}
else if (p[0] == 'P')
{ // POST 方式
    memcpy(httpHeader->method, "POST", 4);
    memcpy(httpHeader->url, &p[5], strlen(p) - 14);
}
printf("正在访问url：%s\n", httpHeader->url);
```

第二行读取host和cookie信息：

```
switch (p[0])
{
case 'H': // Host
    memcpy(httpHeader->host, &p[6], strlen(p) - 6); //将Host复制到httpHeader->host中
    break;
case 'C': // Cookie
    if (strlen(p) > 8)
    {
        char header[8];
        ZeroMemory(header, sizeof(header));
        memcpy(header, p, 6);
        if (!strcmp(header, "Cookie"))
        {
            memcpy(httpHeader->cookie, &p[8], strlen(p) - 8);
        }
    }
    break;
default:
    break;
}
```

c. 连接目标服务器并将将客户端发送的 HTTP 数据报文直接转发给目标服务器：

```
if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host))
{
    //printf("error\n");
    goto error;
}
printf("代理连接主机 %s 成功\n", httpHeader->host);
```

```
ret = send(((ProxyParam*)lpParameter)->serverSocket, Buffer, strlen(Buffer) + 1, 0);
```

d. 接收目标服务器的响应报文，当收到响应报文之后， 将响应报文转发给客户端：

```
ret = send(((ProxyParam*)lpParameter)->clientSocket, Buffer, sizeof(Buffer), 0);
```

e. 错误处理：

处理完成或发生错误时，进入error，代理服务器等待 200 ms 后，关闭该线程，并清理缓存，然后继续接收并处理下一个请求。

```
error:
    printf("关闭套接字\n");
    Sleep(200);
    closesocket(((ProxyParam*)lpParameter)->clientSocket);
    closesocket(((ProxyParam*)lpParameter)->serverSocket);
    delete lpParameter;
    _endthreadex(0);//中止线程
    return 0;
```

2. Cache(附加):

原理:

浏览器向代理服务器发送HTTP请求。代理服务器收到客户端的HTTP请求后，检查本地Cache是否有该请求的URL。

如果没有该URL的缓存，就直接向目标服务器发送HTTP请求，获取对象，将该请求返回的响应缓存下来，存到本地的Cache下。

如果已有目标的缓存，代理服务器则向目标服务器发送一个请求，该请求增加了“If-Modified-Since”字段，通过此字段，告知目标服务器缓存资源最后修改的时间，服务器通过对比最后修改时间来判断缓存是否过期。如果没过期，服务器返回状态码304，代理服务器直接将本地缓存发送给客户端；如果缓存过期，服务器返回状态码200，同时返回一个更新过的响应，代理服务器接收后，将该响应发回给客户端，并更新本地缓存。

实现：

```
//定义缓存变量
char filename[100] = { 0 };
_Post_ _Notnull_ FILE* in;//定义了一个文件指针，用于指向输入文件。_Post_ _Notnull_是一个宏，表示在函数返回之后，in指针必须不为空
char* DateBuffer;//指向日期缓冲区。日期缓冲区用于存储从网络数据包中解析出的日期值
char date_str[30];  //定义了一个字符数组，用于存储日期字符串。日期字符串通常由年、月、日组成，例如 "2021-08-01"
FILE* fp;
```

判断本地cache是否有缓存，即判断能否打开相应的缓存文件：

```
if ((fopen_s(&in, filename, "r")) == 0)
{
    printf("\n有缓存\n");

    getfileDate(in, date_str);//得到本地缓存文件中的日期date_str
    fclose(in);
    //printf("date_str:%s\n", date_str);
    sendnewHTTP(Buffer, date_str);
    //向服务器发送一个请求，该请求需要增加 “If-Modified-Since” 字段
    //服务器通过对比时间来判断缓存是否过期
    haveCache = TRUE;
}
```

如果有，则调用getfileDate函数获取日期：

```
//访问本地文件，获取本地缓存中的日期
void getfileDate(FILE* in, char* tempDate)
{
    char field[5] = "Date";
    //ptr，用于存储strtok_s函数的返回值
    char* p, * ptr, temp[5];//p，用于存储从文件中读取的字符串

    char buffer[MAXSIZE];//存储从文件中读取的字符串
    ZeroMemory(buffer, MAXSIZE);
    fread(buffer, sizeof(char), MAXSIZE, in);//从文件中读取字符串，并将其存储在buffer数组中
    const char* delim = "\r\n";//换行符
    ZeroMemory(temp, 5);
    p = strtok_s(buffer, delim, &ptr);//使用strtok_s函数从buffer数组按行分割字符串，并将第一行存储在p中
    //printf("p = %s\n", p);
    int len = strlen(field) + 2;
    while (p) //如果p指针指向的字符串包含"Date"字符串，则使用memcpy函数将日期信息复制到tempDate数组中，并返回。
    {   //如果p指针指向的字符串不包含"Date"字符串，则继续遍历下一个字符串
        if (strstr(p, field) != NULL) {//调用strstr后指针会指向匹配剩余的第一个字符
            memcpy(tempDate, &p[len], strlen(p) - len);
            return;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
}
```

利用日期改造HTTP报文，添加"If-Modified-Since: "字段（sendnewHTTP函数实现），再根据服务器返回的状态码进行操作：如果为200，则调用storefileCache函数更新本地缓存；如果为304，则调用checkfileCache函数直接从本地获取缓存进行转发：

```
if (haveCache == true) {
    checkfileCache(Buffer, httpHeader->url);
}
if (needCache == true) {
    storefileCache(Buffer, httpHeader->url);
}
```

起初，我们定义haveCache和needCache的布尔值分别为FALSE和TRUE，默认无本地缓存，随着访问调用storefileCache和checkfileCache函数而被改变。

3. 网站过滤(附加)：

原理：设置被过滤的网站，在代理服务器转发HTTP请求之前加入URL判断，若一致则输出“该网站已被屏蔽”，并直接转到错误处理，关闭套接字；若不一致则不进行处理。

```
//在发送报文前进行拦截
if (strcmp(httpHeader->url,INVALID_WEBSITE)==0)
{
    printf("************************************\n");
    printf("----------该网站已被屏蔽------------\n");
    printf("************************************\n");
    goto error;
}
```

```
#define INVALID_WEBSITE "http://http.p2hp.com/"    //网站过滤
```

4. 用户过滤(附加):

原理：类似于网站过滤，只需在设置ProxyServerAddr的IP地址时将INADDR_ANY改为具体的IP地址，就能实现唯一用户访问。

```
ProxyServerAddr.sin_family = AF_INET;
ProxyServerAddr.sin_port = htons(ProxyPort);    //将一个无符号短整型数值转换为TCP/IP网络字节序，即大端模式(big-endian)
//ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;  //INADDR_ANY == 0.0.0.0，表示本机的所有IP
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");  //仅限本机用户可访问
```

5. 网站引导(附加)：

原理：设置源网站和目的网站，和网站过滤一样，我们在代理服务器转发HTTP请求之前加入判断，如果URL指向源网站，则更改httpHeader的host和URL为目的网站的URL，这样http请求就被“掉包”了。

```
////网站引导
if (strstr(httpHeader->url, FISH_WEBSITE_FROM) != NULL) {
    printf("\n====================================\n\n");
    printf("-------------已从源网址：%s 转到 目的网址 ：%s ----------------\n", FISH_WEBSITE_FROM, FISH_WEBSITE_TO);
    memcpy(httpHeader->host, FISH_WEBSITE_HOST, strlen(FISH_WEBSITE_HOST) + 1);
    memcpy(httpHeader->url, FISH_WEBSITE_TO, strlen(FISH_WEBSITE_TO));
}
```

```
#define FISH_WEBSITE_FROM "http://http.p2hp.com/"   //钓鱼网站源网址
#define FISH_WEBSITE_TO "http://jwes.hit.edu.cn/"   //钓鱼网站目的网址
#define FISH_WEBSITE_HOST "jwes.hit.edu.cn"         //钓鱼目的地址的主机名
```

6. 源代码如下：

```
//#include "stdafx.h"
#include <tchar.h>
#include <stdio.h>
```

```
10. #include <Windows.h>
11. #include <process.h>
12. #include <string.h>
13.
14. //静态加入一个 lib 库文件,ws2_32.lib 文件，提供了对以下网络相关 API 的支持，若使用其中的 API，则应该将 ws2_32.lib 加入工程
15. #pragma comment(lib, "Ws2_32.lib")
16.
17. #define MAXSIZE 65507 // 发送数据报文的最大长度
18. #define HTTP_PORT 80  // http 服务器端口
19.
20. #define INVALID_WEBSITE "http://http.p2hp.com/"    //网站过滤
21. #define FISH_WEBSITE_FROM "http://http.p2hp.com/"  //钓鱼网站源网址
22. #define FISH_WEBSITE_TO "http://jwes.hit.edu.cn/"  //钓鱼网站目的网址
23. #define FISH_WEBSITE_HOST "jwes.hit.edu.cn"        //钓鱼目的地址的主机名
24.
25. // Http 重要头部数据
26. struct HttpHeader
27. {
28.     char method[4];         // POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
29.     char url[1024];         // 请求的 url
30.     char host[1024];        // 目标主机
31.     char cookie[1024 * 10]; // cookie
32.     HttpHeader()
33.     {
34.         ZeroMemory(this, sizeof(HttpHeader));
35.     }
36. };
37.
38. BOOL InitSocket();
39. void ParseHttpHead(char* buffer, HttpHeader* httpHeader);
40. BOOL ConnectToServer(SOCKET* serverSocket, char* host);
41. unsigned int __stdcall ProxyThread(LPVOID lpParameter);
42. void getfileDate(FILE* in, char* tempDate);//从文件中读取日期信息，并将日期信息存储在 tempDate 中
43. void sendnewHTTP(char* buffer, char* datestring);//发送一个新的HTTP请求，并将响应存储在 buffer 中
44. void makeFilename(char* url, char* filename);//根据 URL 生成文件名
45. void storefileCache(char* buffer, char* url);//将文件内容存储到缓存中
46. void checkfileCache(char* buffer, char* filename);//检查缓存中是否存在该文件，如果存在，则将文件内容返回
47.
```

```
48.
49. /*
50. 代理相关参数:
51.     SOCKET: 本质是一个 unsigned int 整数，是唯一的 ID
52.     sockaddr_in：用来处理网络通信的地址。sockaddr_in 用于 socket 定义和赋
    值；sockaddr 用于函数参数
53.         short   sin_family;         地址族
54.         u_short sin_port;           16 位 TCP/UDP 端口号
55.         struct  in_addr sin_addr;   32 位 IP 地址
56.         char    sin_zero[8];        不使用
57. */
58. SOCKET ProxyServer;
59. sockaddr_in ProxyServerAddr;
60. const int ProxyPort = 10240;
61.
62. //缓存相关参数
63. boolean haveCache = FALSE;
64. boolean needCache = TRUE;
65.
66. // 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
67. // 可以使用线程池技术提高服务器效率
68. // const int ProxyThreadMaxNum = 20;
69. // HANDLE ProxyThreadHandle[ProxyThreadMaxNum] = {0};
70. // DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};
71.
72. struct ProxyParam
73. {
74.     SOCKET clientSocket;
75.     SOCKET serverSocket;
76. };
77.
78. int _tmain(int argc, _TCHAR* argv[])
79. {
80.     printf("代理服务器正在启动\n");
81.     printf("初始化...\n");
82.     if (!InitSocket())
83.     {
84.         printf("socket 初始化失败\n");
85.         return -1;
86.     }
87.     printf("代理服务器正在运行，监听端口 %d\n", ProxyPort);
88.     SOCKET acceptSocket = INVALID_SOCKET; //无效套接字-->初始化？
89.     ProxyParam* lpProxyParam;
90.     HANDLE hThread; //句柄：和对象一一对应的 32 位无符号整数值
```

```
91.     DWORD dwThreadID; //unsigned long
92.
93.     // 代理服务器不断监听
94.     while (true)
95.     {
96.         //accept 将客户端的信息绑定到一个 socket 上，也就是给客户端创建一个 socket，通过返回值返回给我们客户端的 socket
97.         acceptSocket = accept(ProxyServer, NULL, NULL);
98.         lpProxyParam = new ProxyParam;
99.         if (lpProxyParam == NULL)
100.            {
101.                continue;
102.            }
103.            lpProxyParam->clientSocket = acceptSocket;
104.
105.            //_beginthreadex 创建线程，第 3、4 个参数分别为线程执行函数、线程函数的参数
106.            hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0);
107.
108.            /* CloseHandle
109.                只是关闭了一个线程句柄对象，表示我不再使用该句柄，
110.                即不对这个句柄对应的线程做任何干预了，和结束线程没有一点关系
111.            */
112.            CloseHandle(hThread);
113.
114.            //延迟 from <windows.h>
115.            Sleep(200);
116.        }
117.        closesocket(ProxyServer);
118.        WSACleanup();
119.        return 0;
120.    }
121.
122.    //************************************
123.    // Method: InitSocket
124.    // FullName: InitSocket
125.    // Access: public
126.    // Returns: BOOL
127.    // Qualifier: 初始化套接字
128.    //************************************
129.    BOOL InitSocket()
130.    {
```

```
131.
132.        // 加载套接字库（必须）
133.        WORD wVersionRequested;
134.        WSADATA wsaData;
135.        // 套接字加载时错误提示
136.        int err;
137.        // 版本 2.2
138.        wVersionRequested = MAKEWORD(2, 2);
139.        // 加载 dll 文件 Scoket 库
140.        err = WSAStartup(wVersionRequested, &wsaData);
141.        if (err != 0)
142.        {
143.            // 找不到 winsock.dll
144.            printf("加载 winsock 失败，错误代码
    为: %d\n", WSAGetLastError());
145.            return FALSE;
146.        }
147.        if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion
    ) != 2)
148.        {
149.            printf("不能找到正确的 winsock 版本\n");
150.            WSACleanup();
151.            return FALSE;
152.        }
153.
154.        /*socket() 函数:
155.            int af: 地址族规范。当前支持的值为 AF_INET 或 AF_INET6，这是
    IPv4 和 IPv6 的 Internet 地址族格式
156.            int type: 新套接字的类型规范。SOCK_STREAM 1 是一种套接字类
    型，可通过 OOB 数据传输机制提供
157.                        顺序的，可靠的，双向的，基于连接的字节流
158.            int protocol: 协议。值为 0，则调用者不希望指定协议，服务提供
    商将选择要使用的协议
159.        */
160.        ProxyServer = socket(AF_INET, SOCK_STREAM, 0);
161.
162.        if (INVALID_SOCKET == ProxyServer)
163.        {
164.            printf("创建套接字失败，错误代码
    为:  %d\n", WSAGetLastError());
165.            return FALSE;
166.        }
167.
168.        ProxyServerAddr.sin_family = AF_INET;
```

```
169.        ProxyServerAddr.sin_port = htons(ProxyPort);    //将一个无符
    号短整型数值转换为TCP/IP 网络字节序，即大端模式(big-endian)
170.        //ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;  //INAD
    DR_ANY == 0.0.0.0，表示本机的所有IP
171.        ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"
    );  //仅限本机用户可访问
172.
173.
174.        //绑定套接字与网络地址
175.        if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SO
    CKADDR)) == SOCKET_ERROR)
176.        {
177.            printf("绑定套接字失败\n");
178.            return FALSE;
179.        }
180.
181.        //监听套接字
182.        if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR)
183.        {
184.            printf("监听端口%d 失败", ProxyPort);
185.            return FALSE;
186.        }
187.        return TRUE;
188.    }
189.
190.    //************************************
191.    // Method: ProxyThread
192.    // FullName: ProxyThread
193.    // Access: public
194.    // Returns: unsigned int __stdcall
195.    // Qualifier: 线程执行函数
196.    // Parameter: LPVOID lpParameter
197.    // 线程的生命周期就是线程函数从开始执行到线程结束
198.    // //************************************
199.    unsigned int __stdcall ProxyThread(LPVOID lpParameter)
200.    {
201.        //定义缓存变量
202.        char filename[100] = { 0 };
203.        _Post_ _Notnull_ FILE* in;//定义了一个文件指针，用于指向输入文
    件。_Post_ _Notnull_是一个宏，表示在函数返回之后，in 指针必须不为空
204.        char* DateBuffer;//指向日期缓冲区。日期缓冲区用于存储从网络数据
    包中解析出的日期值
205.        char date_str[30];  //定义了一个字符数组，用于存储日期字符串。日
    期字符串通常由年、月、日组成，例如 "2021-08-01"
```

```
206.        FILE* fp;
207.
208.        char Buffer[MAXSIZE];
209.        char* CacheBuffer;
210.        ZeroMemory(Buffer, MAXSIZE);
211.        SOCKADDR_IN clientAddr;
212.        int length = sizeof(SOCKADDR_IN);
213.        int recvSize;
214.        int ret;
215.        recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Bu
    ffer, MAXSIZE, 0);
216.        HttpHeader* httpHeader = new HttpHeader();
217.        CacheBuffer = new char[recvSize + 1];
218.
219.        //goto 语句不能跳过实例化（局部变量定义），把实例化移到函数开头就
    可以了
220.        if (recvSize <= 0)
221.        {
222.            goto error;
223.        }
224.
225.        ZeroMemory(CacheBuffer, recvSize + 1);
226.        memcpy(CacheBuffer, Buffer, recvSize);  //由 src 指向地址为起始
    地址的连续 n 个字节的数据复制到以 destin 指向地址为起始地址的空间内
227.        //解析 httpheader
228.        ParseHttpHead(CacheBuffer, httpHeader);
229.
230.        ZeroMemory(date_str, 30);
231.        printf("httpHeader->url : %s\n", httpHeader->url);
232.        makeFilename(httpHeader->url, filename);
233.        //printf("filename 是 %s\n", filename);
234.        if ((fopen_s(&in, filename, "r")) == 0)
235.        {
236.            printf("\n 有缓存\n");
237.
238.            getfileDate(in, date_str);//得到本地缓存文件中的日期
    date_str
239.            fclose(in);
240.            //printf("date_str:%s\n", date_str);
241.            sendnewHTTP(Buffer, date_str);
242.            //向服务器发送一个请求，该请求需要增加“If-Modified-
    Since”字段
243.            //服务器通过对比时间来判断缓存是否过期
244.            haveCache = TRUE;
```

```
245.        }
246.
247.        delete CacheBuffer;
248.        //printf("test\n");
249.
250.        //在发送报文前进行拦截
251.        if (strcmp(httpHeader->url,INVALID_WEBSITE)==0)
252.        {
253.            printf("************************************\n");
254.            printf("----------该网站已被屏蔽------------\n");
255.            printf("************************************\n");
256.            goto error;
257.        }
258.
259.        ////网站引导
260.        if (strstr(httpHeader->url, FISH_WEBSITE_FROM) != NULL) {
261.            printf("\n====================================\n\n");
262.            printf("-------------已从源网址：%s 转到 目的网址 ： %s ----
    ------------\n", FISH_WEBSITE_FROM, FISH_WEBSITE_TO);
263.            memcpy(httpHeader->host, FISH_WEBSITE_HOST, strlen(FISH_
    WEBSITE_HOST) + 1);
264.            memcpy(httpHeader->url, FISH_WEBSITE_TO, strlen(FISH_WEB
    SITE_TO));
265.         }
266.
267.
268.        if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSock
    et, httpHeader->host))
269.        {
270.            //printf("error\n");
271.            goto error;
272.        }
273.        printf("代理连接主机 %s 成功\n", httpHeader->host);
274.        //printf("test");
275.
276.        // 将客户端发送的 HTTP 数据报文直接转发给目标服务器
277.        ret = send(((ProxyParam*)lpParameter)->serverSocket, Buffer,
     strlen(Buffer) + 1, 0);
278.
279.        // 等待目标服务器返回数据
280.        recvSize = recv(((ProxyParam*)lpParameter)->serverSocket, Bu
    ffer, MAXSIZE, 0);
281.        if (recvSize <= 0)
282.        {
```

```
283.            goto error;
284.        }
285.        if (haveCache == true) {
286.            checkfileCache(Buffer, httpHeader->url);
287.        }
288.        if (needCache == true) {
289.            storefileCache(Buffer, httpHeader->url);
290.        }
291.
292.        // 将目标服务器返回的数据直接转发给客户端
293.        ret = send(((ProxyParam*)lpParameter)->clientSocket, Buffer,
    sizeof(Buffer), 0);
294.
295.
296.    // 错误处理
297.    error:
298.        printf("关闭套接字\n");
299.        Sleep(200);
300.        closesocket(((ProxyParam*)lpParameter)->clientSocket);
301.        closesocket(((ProxyParam*)lpParameter)->serverSocket);
302.        delete lpParameter;
303.        _endthreadex(0);//中止线程
304.        return 0;
305.    }
306.
307.    //*************************************
308.    // Method: ParseHttpHead
309.    // FullName: ParseHttpHead
310.    // Access: public
311.    // Returns: void
312.    // Qualifier: 解析 TCP 报文中的 HTTP 头部
313.    // Parameter: char * buffer
314.    // Parameter: HttpHeader * httpHeader
315.    //*************************************
316.    void ParseHttpHead(char* buffer, HttpHeader* httpHeader)
317.    {
318.        char* p;
319.        char* ptr;
320.        const char* delim = "\r\n";
321.        p = strtok_s(buffer, delim, &ptr); // 提取第一行
322.        printf("%s\n", p);
323.        if (p[0] == 'G')
324.        { // GET 方式
325.            memcpy(httpHeader->method, "GET", 3);
```

```
326.            memcpy(httpHeader->url, &p[4], strlen(p) - 13);
327.        }
328.        else if (p[0] == 'P')
329.        { // POST 方式
330.            memcpy(httpHeader->method, "POST", 4);
331.            memcpy(httpHeader->url, &p[5], strlen(p) - 14);
332.        }
333.        printf("正在访问 url: %s\n", httpHeader->url);
334.        p = strtok_s(NULL, delim, &ptr);    //提取第二行
335.        while (p)
336.        {
337.            switch (p[0])
338.            {
339.            case 'H': // Host
340.                memcpy(httpHeader->host, &p[6], strlen(p) - 6); //将
    Host 复制到 httpHeader->host 中
341.                break;
342.            case 'C': // Cookie
343.                if (strlen(p) > 8)
344.                {
345.                    char header[8];
346.                    ZeroMemory(header, sizeof(header));
347.                    memcpy(header, p, 6);
348.                    if (!strcmp(header, "Cookie"))
349.                    {
350.                        memcpy(httpHeader->cookie, &p[8], strlen(p)
    - 8);
351.                    }
352.                }
353.                break;
354.            default:
355.                break;
356.            }
357.            p = strtok_s(NULL, delim, &ptr); //读取下一行
358.        }
359.    }
360.
361.    //************************************
362.    // Method: ConnectToServer
363.    // FullName: ConnectToServer
364.    // Access: public
365.    // Returns: BOOL
366.    // Qualifier: 根据主机创建目标服务器套接字，并连接
367.    // Parameter: SOCKET * serverSocket
```

```
368.    // Parameter: char * host
369.    //***********************************
370.    BOOL ConnectToServer(SOCKET* serverSocket, char* host)
371.    {
372.        sockaddr_in serverAddr;
373.        serverAddr.sin_family = AF_INET;
374.        serverAddr.sin_port = htons(HTTP_PORT);
375.        HOSTENT* hostent = gethostbyname(host);
376.        if (!hostent)
377.        {
378.            //printf("error_hostent\n");
379.            return FALSE;
380.        }
381.        in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
382.        serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
383.        *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
384.        if (*serverSocket == INVALID_SOCKET)
385.        {
386.            //printf("error_serverSocket\n");
387.            return FALSE;
388.        }
389.        if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(se
    rverAddr)) == SOCKET_ERROR)
390.        {
391.            //printf("error_connect\n");
392.            closesocket(*serverSocket);
393.            return FALSE;
394.        }
395.        return TRUE;
396.    }
397.
398.    //访问本地文件，获取本地缓存中的日期
399.    void getfileDate(FILE* in, char* tempDate)
400.    {
401.        char field[5] = "Date";
402.        //ptr，用于存储 strtok_s 函数的返回值
403.        char* p, * ptr, temp[5];//p，用于存储从文件中读取的字符串
404.
405.        char buffer[MAXSIZE];//存储从文件中读取的字符串
406.        ZeroMemory(buffer, MAXSIZE);
407.        fread(buffer, sizeof(char), MAXSIZE, in);//从文件中读取字符
    串，并将其存储在 buffer 数组中
408.        const char* delim = "\r\n";//换行符
409.        ZeroMemory(temp, 5);
```

```
410.        p = strtok_s(buffer, delim, &ptr);//使用 strtok_s 函数从 buffer 数组按行分割字符串，并将第一行存储在 p 中
411.        //printf("p = %s\n", p);
412.        int len = strlen(field) + 2;
413.        while (p) //如果 p 指针指向的字符串包含"Date"字符串，则使用 memcpy 函数将日期信息复制到 tempDate 数组中，并返回。
414.            //如果 p 指针指向的字符串不包含"Date"字符串，则继续遍历下一个字符串
415.        {
416.            if (strstr(p, field) != NULL) {//调用 strstr 后指针会指向匹配剩余的第一个字符
417.                memcpy(tempDate, &p[len], strlen(p) - len);
418.                return;
419.            }
420.            p = strtok_s(NULL, delim, &ptr);
421.        }
422.    }
423.
424.    //改造 HTTP 请求报文
425.    void sendnewHTTP(char* buffer, char* datestring) {
426.        const char* field = "Host";
427.        const char* newfield = "If-Modified-Since: ";//分别用于表示请求报文段中的 Host 字段和要插入的新字段
428.        //const char *delim = "\r\n";
429.        char temp[MAXSIZE];//存储插入新字段后的请求报文
430.        ZeroMemory(temp, MAXSIZE);
431.        char* pos = strstr(buffer, field);//获取请求报文段中 Host 后的部分信息
432.        int i = 0;
433.        for (i = 0; i < strlen(pos); i++) {
434.            temp[i] = pos[i];//将 pos 复制给 temp
435.        }
436.        *pos = '\0';
437.        //将 pos 指针指向 Host 字段后的第一个字符，然后遍历新字段，将新字段中的每个字符插入到 pos 指针指向的位置
438.        while (*newfield != '\0') {  //插入 If-Modified-Since 字段
439.            *pos++ = *newfield++;
440.        }
441.        while (*datestring != '\0') {//插入对象文件的最新被修改时间
442.            *pos++ = *datestring++;
443.        }
444.        *pos++ = '\r';//符合报文格式
445.        *pos++ = '\n';
```

```
446.        for (i = 0; i < strlen(temp); i++)//将原始请求报文段中的剩余字符复制到新字段之后
447.        {
448.            *pos++ = temp[i];
449.        }
450.    }
451.
452.    //根据 url 构造文件名
453.    void makeFilename(char* url, char* filename) {
454.        while (*url != '\0') {
455.            if ('a' <= *url && *url <= 'z') {
456.                *filename++ = *url;//如果当前字符在'a'到'z'的范围内，将其复制到文件名字符串中
457.            }
458.            url++;
459.        }
460.        strcat_s(filename, strlen(filename) + 9, ".txt");
461.    }
462.
463.    //检测服务器返回的状态码，如果是 200 则把数据进行本地更新缓存
464.    void storefileCache(char* buffer, char* url) {
465.        char* p, * ptr, tempBuffer[MAXSIZE + 1];
466.
467.        const char* delim = "\r\n";
468.        ZeroMemory(tempBuffer, MAXSIZE + 1);
469.        memcpy(tempBuffer, buffer, strlen(buffer));
470.        p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
471.
472.        if (strstr(tempBuffer, "200") != NULL) {  //状态码是 200 时缓存
473.            char filename[100] = { 0 };
474.            makeFilename(url, filename);
475.            printf("filename : %s\n", filename);
476.            FILE* out;
477.            fopen_s(&out, filename, "w+");
478.            fwrite(buffer, sizeof(char), strlen(buffer), out);//使用 fopen_s 函数以写入模式打开文件，并将响应内容写入文件
479.            fclose(out);
480.            printf("\n==================更新缓存ok==================\n");
481.        }
482.    }
483.
484.    //检测服务器返回的状态码，如果是 304 则从本地获取缓存进行转发，否则需要更新缓存
```

```
485.    void checkfileCache(char* buffer, char* filename)
486.    {
487.        char* p, * ptr, tempBuffer[MAXSIZE + 1];
488.        const char* delim = "\r\n";
489.        ZeroMemory(tempBuffer, MAXSIZE + 1);
490.        memcpy(tempBuffer, buffer, strlen(buffer));//将 buffer 复制到
    tempBuffer 中
491.        p = strtok_s(tempBuffer, delim, &ptr);//提取状态码所在行
492.        //主机返回的报文中的状态码为 304 时返回已缓存的内容
493.        if (strstr(p, "304") != NULL) {
494.            printf("\n=================从本机获得缓存
    ====================\n");
495.            ZeroMemory(buffer, strlen(buffer));
496.            FILE* in = NULL;
497.            if ((fopen_s(&in, filename, "r")) == 0) {
498.                fread(buffer, sizeof(char), MAXSIZE, in);//使用
    fopen_s 函数以读取模式打开文件，并将文件内容存储在 buffer 数组中
499.                fclose(in);
500.            }
501.            needCache = FALSE;
502.        }
503.    }
```

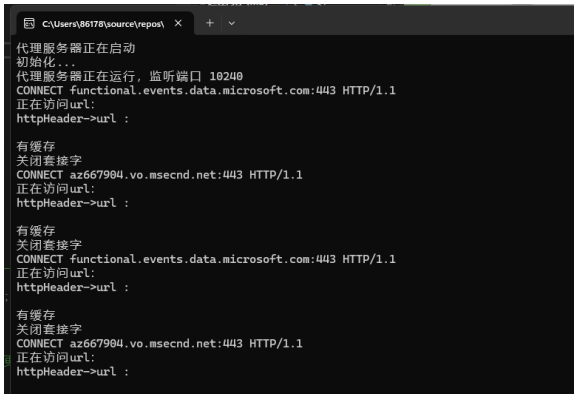
实验结果：

1）基本功能和cache：

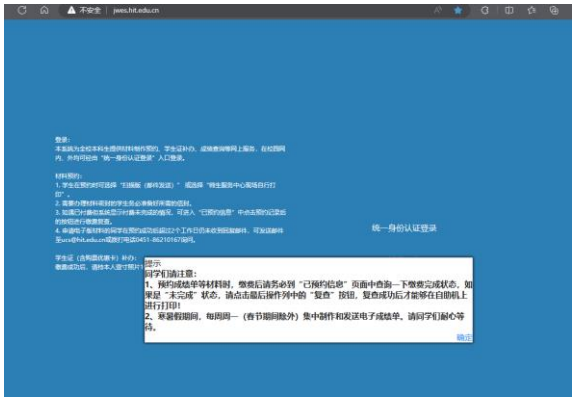
首先设置浏览器的代理服务器，IP为127.0.0.1，端口号为10240

此时代理服务器还未开启，刷新浏览器发现无法连接：

然后启动代理服务器：



打开一个http网站，已经可以访问，并且创建缓存文件：




```
GET http://jwes.hit.edu.cn/resources/images/common/butt.png HTTP/1.1
正在访问url: http://jwes.hit.edu.cn/resources/images/common/butt.png
httpHeader->url : http://jwes.hit.edu.cn/resources/images/common/butt.png
代理连接主机 jwes.hit.edu.cn 成功
filename : httpjweshiteducnresourcesimagescommonbuttpng.txt

==================更新缓存ok==================
关闭套接字
```

刷新该网站，此时获得的是本机的缓存：

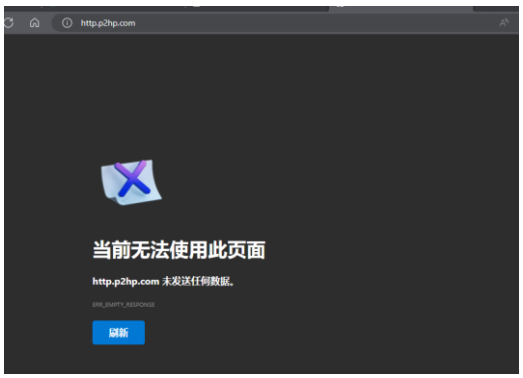
```
GET http://jwes.hit.edu.cn/resources/css/common/style1.css HTTP/1.1
正在访问url: http://jwes.hit.edu.cn/resources/css/common/style1.css
httpHeader->url : http://jwes.hit.edu.cn/resources/css/common/style1.css

有缓存
代理连接主机 jwes.hit.edu.cn 成功

==================从本机获得缓存====================
关闭套接字
```

2）网站过滤：

以http://http.p2hp.com/为例，此时该网站已经被拦截，无法访问。




```
GET http://http.p2hp.com/ HTTP/1.1
正在访问url: http://http.p2hp.com/
httpHeader->url : http://http.p2hp.com/
************************************
----------该网站已被屏蔽------------
************************************
关闭套接字
```

3）用户过滤：

在代码中将用户127.0.0.1过滤，当我们再次访问http://http.p2hp.com/结果如下：


```
获取用户IP地址：127.0.0.1
IP被禁用
关闭套接字
```

4）网站引导：

此处我们将网站http://http.p2hp.com重定向到我们学校的网站http://jwes.hit.edu.cn/：

可以看到地址栏的网址没有变化，但显示的内容确实我们学校网站的内容。
至此所有功能展示完毕。

问题讨论：

该代理服务器只对使用http的网站起作用，但是如今http正在慢慢退出舞台，越来越多网站使用或换成了https协议。要实现对https的代理需要更多的东西，这些东西我还不太清楚，还需要进一步的学习。

心得体会：

这是一次难度合适且十分充实的实验。虽然之前我也在用着代理服务器，但对其中的实现知之甚少。通过这次实验，我学到了http代理服务器的工作原理，并通过附加实验，实现了如钓鱼网站这样一些比较好玩的功能。同时明白了在代理服务器中加Cache的功能能让访问更快的这种设计思路。