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 | 刘天瑞 | | 院系 | 未来技术学院 | | |
| 班级 | 20W0361 | | 学号 | 7203610121 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | G001 | | 实验时间 | 2022.10.5 | | |
| 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：

（注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）

本次实验的主要目的。

答：

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：

概述本次实验的主要内容，包含的实验项等。

答：

(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。（选作内容，加分项目，可以当堂完成或课下完成）

(3) 扩展HTTP代理服务器，支持如下功能：（选作内容，加分项目，可以当堂完成或课下完成）

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。

答：

## 1.了解实验相关基础知识

### i）代理服务器的概念

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。普通Web应用通信方式与采用代理服务器的通信方式的对比如下图所示：

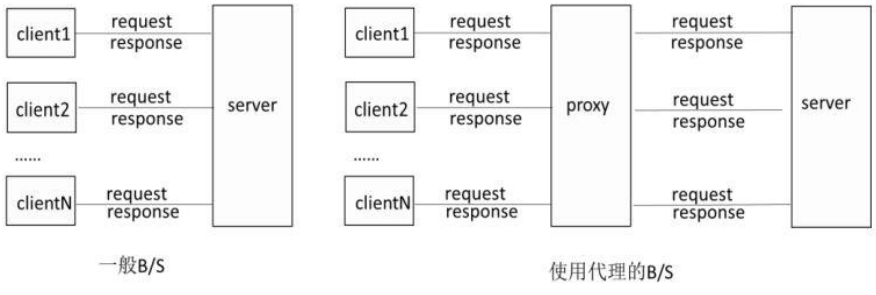



图1 普通Web应用与采用代理服务器的通信方式对比

代理服务器可以认为是TCP/IP网络应用的客户端和服务器端的结合。一方面，它是浏览器客户端的服务器端，另一方面，它也是目标服务器的客户端。浏览器将请求报文发送给代理服务器，代理服务器经过一些处理或者不经过处理，将请求报文转发给目标服务器；目标服务器相应请求报文发出响应报文，代理服务器接受到响应报文之后直接

将响应报文转发给浏览器客户端。

针对本实验的特定要求，概括我们所要实现的代理服务器功能如下：

代理服务器在指定端口（本实验中所指定的是10240端口）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），代理服务器接收到浏览器对远程网站的浏览请求时，首先会查看浏览器来源的IP地址，如果属于被限制的用户，则认为没有接受到访问请求（用户过滤功能）。否则，查看其请求的host主机，如果属于不允许访问的主机，则默认不向目标服务器发送请求（网站过滤功能）；如果属于被引导的网站，则对该网站的请求报文中的host主机地址和url进行更改（网站引导功能）。而对于Cache功能的实现，基本可以概括为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），若找到对象文件，则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。若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。

除此之外，本实验要设计的服务器属于多用户代理服务器。首先，代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，创建一个子线程，由子线程执行上述一对一的代理过程，服务结束后子线程终止。与此同时，主线程继续接受下一个客户的代理服务。

HTTP代理服务器的流程图如下：

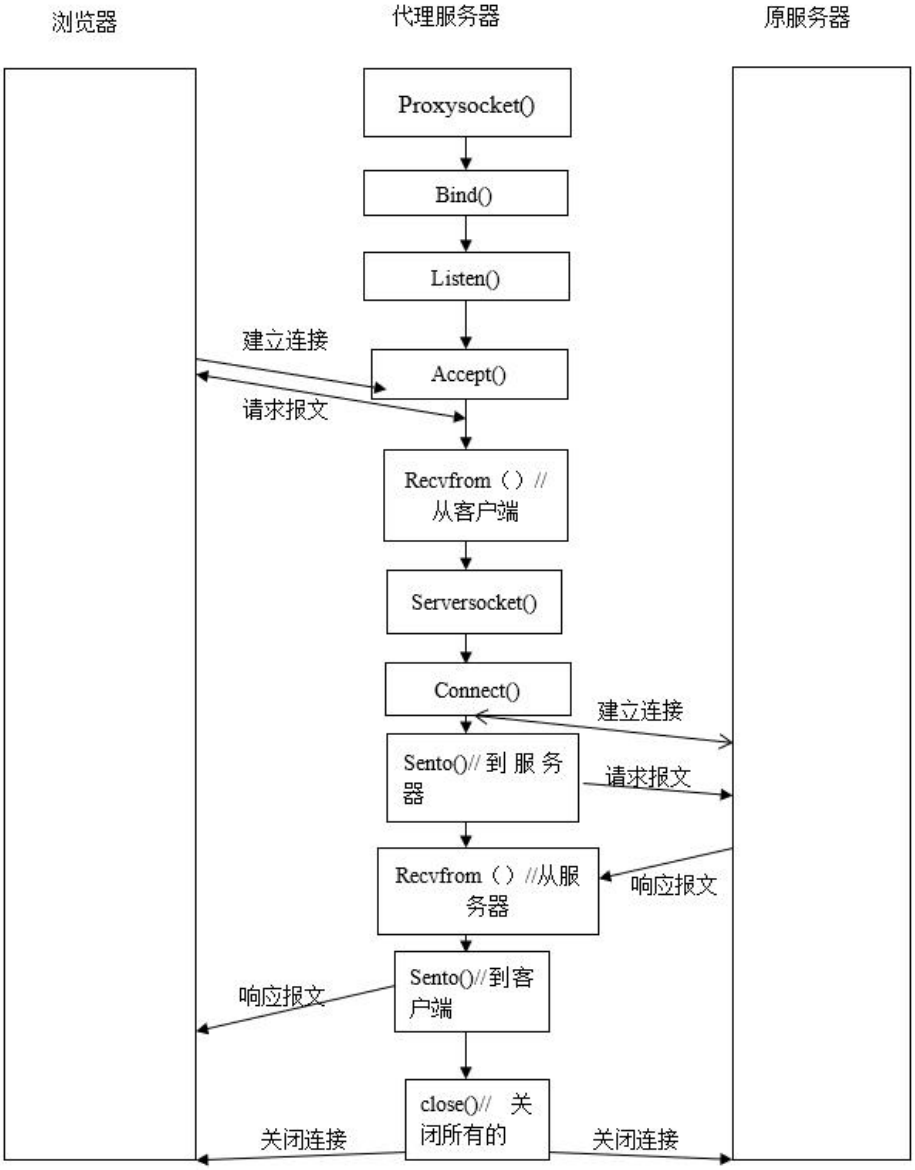



图2　HTTP代理服务器流程图

**ii）TCP客户端与软件端的流程**

(a)　TCP客户端软件流程

根据上课的PPT可以总结为：

1. 根据目标服务器IP地址与端口号创建套接字（socket）
2. 连接服务器（connect）：三次握手
3. 发送请求报文（send）
4. 接收返回报文（recv），返回3或者5
5. 关闭连接（closesocket）

(b)　TCP服务器端软件流程

根据上课的PPT，可以总结为：

1. 创建套接字（socket），绑定套接字的本地IP地址和端口号（bind），然后转到监听模式并设置连接请求队列大小（listen）。

2. 从连接请求队列中取出一个连接请求，并同意连接（accept）。在TCP连接过程中进行了三次握手。

3. 收到请求报文（recv）

4. 发送数据（send）返回3或者5

5. 关闭连接（closesocket）返回2

(c) TCP软件端与服务器端流程图

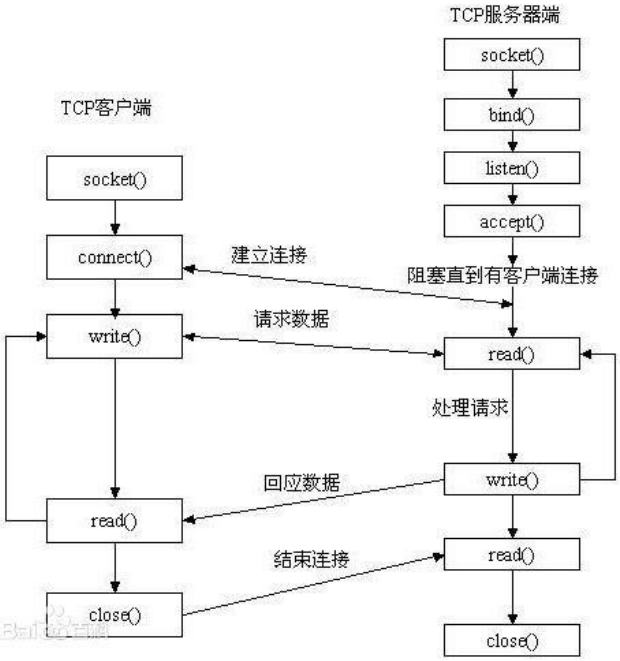
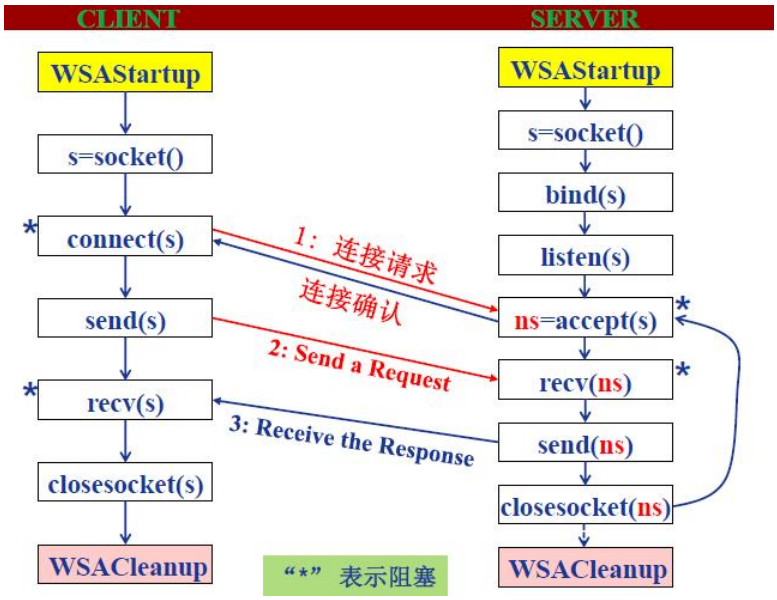



图3 TCP软件端与服务器端流程图

**iii) 网络应用的Socket API(TCP)调用基本流程**

根据上课的PPT，可以总结为：




**iv) 常用C Socket编程函数及其功能**

根据上课的PPT，可以总结为：


## 2.修改程序，完成基本功能

对于代理服务器的实现，我采用基于c++语言示例代码修改的方式完成，使用的编程环境为visual studio 2022，程序初始提供的示例代码部分地方无法使用，因此，经过查询资料，发现有以下地方需要修改：

**i）去掉第一行的#include "stdafx.h"；**

**ii）goto语句后不能再定义变量。**

## 3.程序基本功能描述

在修改完成后，示例代码便可正常运行，以下简述示例代码的流程与功能：

1）初始化一个套接字，利用 bind() 函数将该套接字与服务器 host 地址绑定，地址设为 “127.0.0.1”；同时也要绑定端口号，在示例代码中，端口号被设置为“10240”。在设置完毕后，利用 listen() 函数对该端口进行监听。

2）通过设置 accept() 函数，对每个到来的请求进行接收和相应，为提高效率，对每个请求，代理服务器都创建一个新的线程来处理。

3）利用 recv() 和 send() 函数，接收来自客户端的 HTTP 请求，并通过这个代理服务器将该请求转发给服务器；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。在这里，代理服务器相当于一个中介，提供一个代理的服务，所有的请求和响应都经过它。

4）处理完成后，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。对于客户端而言，它只要将正常发送的请求发给代理服务器，就可以接收到对应的响应。

## 4.拓展HTTP代理服务器（网站过滤，用户过滤，网站引导）

设置被屏蔽的网站为哈工大电子邮件系统（http://mail.hit.edu.cn/），设置钓鱼网站为由进入哈工大网站（http://today.hit.edu.cn/）重定向至教学管理服务系统（http://jwts.hit.edu.cn/），相关示例网站可以更换（例如哈工大教务处网站：http://jwc.hit.edu.cn/），但需要注意，更换的示例网站必须采用HTTP协议，而非HTTPS协议。

### 1) 网站过滤

在线程执行函数ProxyThread（）中，解析TCP报文中的HTTP头部，将HTTP头部中的URL与对应屏蔽网站进行比较，如果相同，则打印相关语句并直接跳转到error，从而实现网站过滤功能：

```
350     //网站屏蔽
351     if (strcmp(httpHeader->url, "http://mail.hit.edu.cn/") == 0) {
352         printf("\n======================================\n\n");
353         printf("您所前往的网站已被屏蔽！\n");
354         goto error;
```

图4 网站屏蔽相关代码

### 2) 用户过滤

修改strcmp(ip, "127.0.0.1") == 0改为strcmp(ip, "127.0.0.1") == 1表示只允许本机访问代理服务器，从而实现用户过滤功能。代码如下：

```
325     //用户过滤
326     char hostname[128];
327     int retnew = gethostname(hostname, sizeof(hostname));
328     HOSTENT *hent = gethostbyname(hostname);
329     char *ip = inet_ntoa(*(in_addr*)*hent->h_addr_list);  //获取本地ip地址
330     if (strcmp(ip, "127.0.0.1") == 0) {
331         printf("\n==================================\n\n");
332         printf("客户ip地址：%s\n", ip);
333         printf("您的主机已被屏蔽！\n");
334         goto error;
335     }
```

图5 用户过滤相关代码

**3) 网站引导（钓鱼）**

网站引导功能同样在线程执行函数ProxyThread中实现。实现思路为：比较HTTP头部中的URL字段是否与重定向源网址相同，若相同，则说明客户端想要访问的网址是重定向源网址，需要执行重定向操作。具体而言，在重定向操作中，我们分部分构造302报文，将其中的IP地址修改为钓鱼目的IP地址，在最后，将修改好的302报文通过send函数返回给客户端。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：

网站引导的核心函数代码如下图所示：

```
357     //网站引导：钓鱼网站
358     if (strcmp(httpHeader->url, "http://today.hit.edu.cn/") == 0) {
359         printf("\n==================================\n\n");
360         printf("钓鱼成功：您所前往的http://today.hit.edu.cn/已被引导至http://jwts.hit.edu.cn\n");
361         memcpy(httpHeader->host, "jwts.hit.edu.cn", 22);
```

图6 网站引导相关代码

## 5.实现Cache功能

在本次HTTP代理服务器的相关实验中，Cache的基本功能如下：

代理服务器第一次和客户端通信时，会保留Cache；当客户端再次请求本地存在的cache页面时，代理服务器会通过If-Modified-Since头将先前目标服务器端发过来的Last-Modified最后修改时间戳发送回去，让目标服务器端进行验证，通过这个时间戳判断客户端的页面是否是最新的，如果状态码为200，则表示内容不是最新的，则由目标服务器返回新的内容，如果内容未经修改，仍是最新的，则返回304告诉客户端其本地cache的页面是最新的，于是代理服务器可以本地Cache的发送给客户端。

具体到代码实现：首先通过修改后的ParseHttpHead（）函数解析TCP报文中的HTTP头部，并在解析的同时判断HTTP头中包含的url是否已经存在于Cache中，若是则会将该url存入代理服务器的Cache中。如果代理服务器的Cache没有了空间，则覆盖Cache的第一个位置。ParseHttpHead（）函数的返回值Have_cache标志着请求的页面在代理服务器上是否有缓存，若是则构造缓存的报文头，由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>，然后将客户端发送的HTTP数据报文直接转发给目标服务器，并等待目标服务器返回数据。在目标服务器返回数据后，由代理服务器解析包含缓存信息的HTTP报文头，读取返回的状态字与页面的最后修改时间。若状态码为304，则说明页面未被修改，打印相关信息后直接由代理服务器将缓存数据转发给客户端；若状态码为200，则表示文件已被修改，首先修改缓存内容，然后再将目标服务器返回的数据直接转发给客户端。若是代理服务器根本没有缓存过该页面，则将该页面缓存到Cache中，然后将客户端发送的HTTP数据报文直接转发给目标服务器，再将目标服务器返回的数据直接转发给客户端。在此过程中，任何一步出现了错误，都直接通过goto语句跳转到error进行

关闭套接字，结束该线程的处理。

由于相关源代码比较多，无法详细在一页中展示，故此处暂不对该部分代码截图，详情请见源代码附录。

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

答：

## 1. 修改Microsoft Edge Web浏览器的代理设置

按照实验指导书修改Microsoft Edge Web浏览器设置如下：

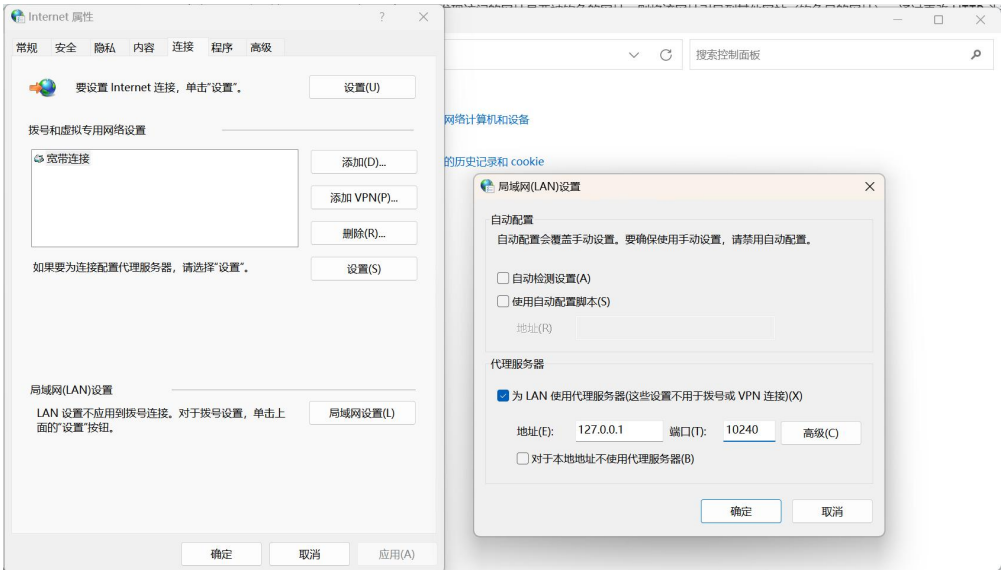



图7 局域网（LAN）设置代理服务器

## 2. 通过代码实现静态链接，并运行程序，实现基础功能

直接运行主程序，然后打开Microsoft Edge Web浏览器，打开一个使用http协议的网站，例如：http://jwc.hit.edu.cn/，可以看到如下结果：

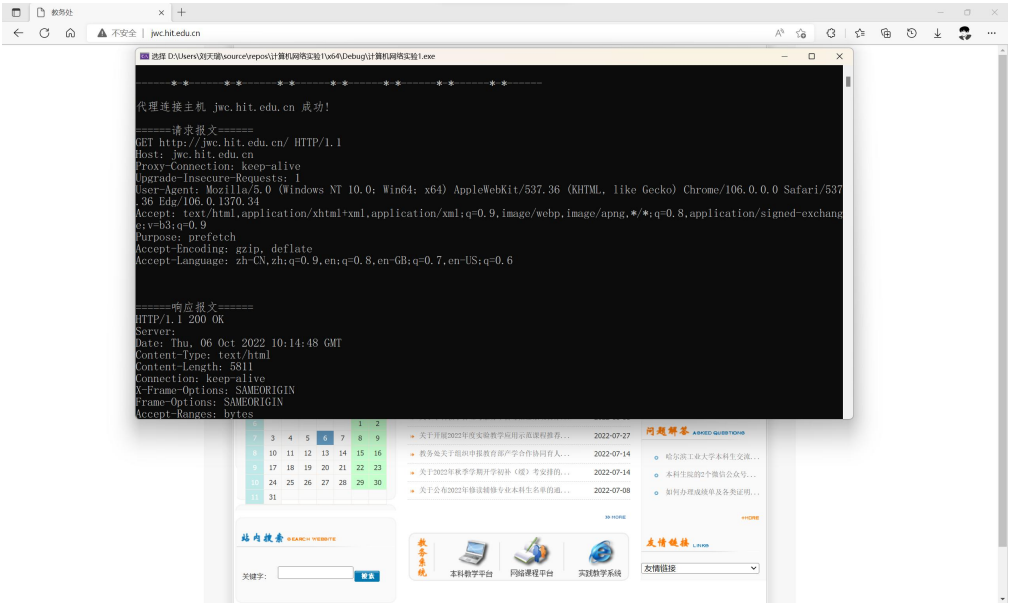



图8 实现网址静态链接基础功能

这说明设置好的代理服务器成功接收了来自客户的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客户进行浏览。

## 3. 验证代理服务器的Cache功能

再次在Microsoft Edge Web浏览器地址栏输入http://jwc.hit.edu.cn/进行访问，结果如下：

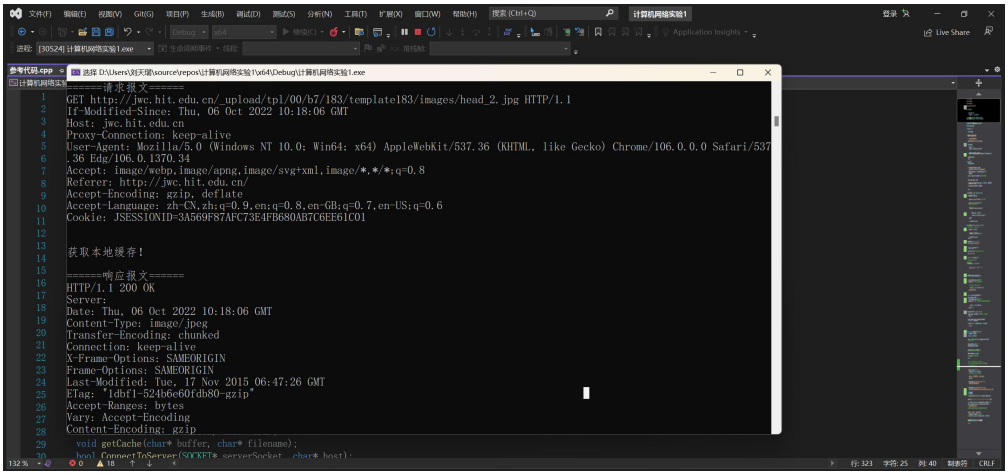



图9 显示获取本地缓存网址

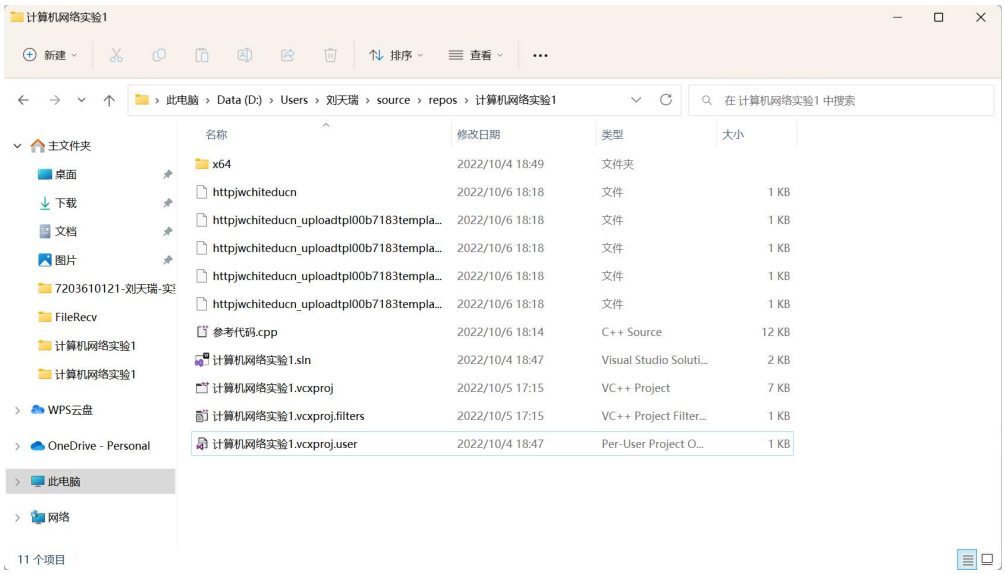



图10 检验浏览过的网址已缓存至目标文件夹

## 4. 验证代理服务器的HTTP拓展功能

### 1) 网站过滤

设定的屏蔽网站为http://mail.hit.edu.cn/，输入网址尝试进行访问：

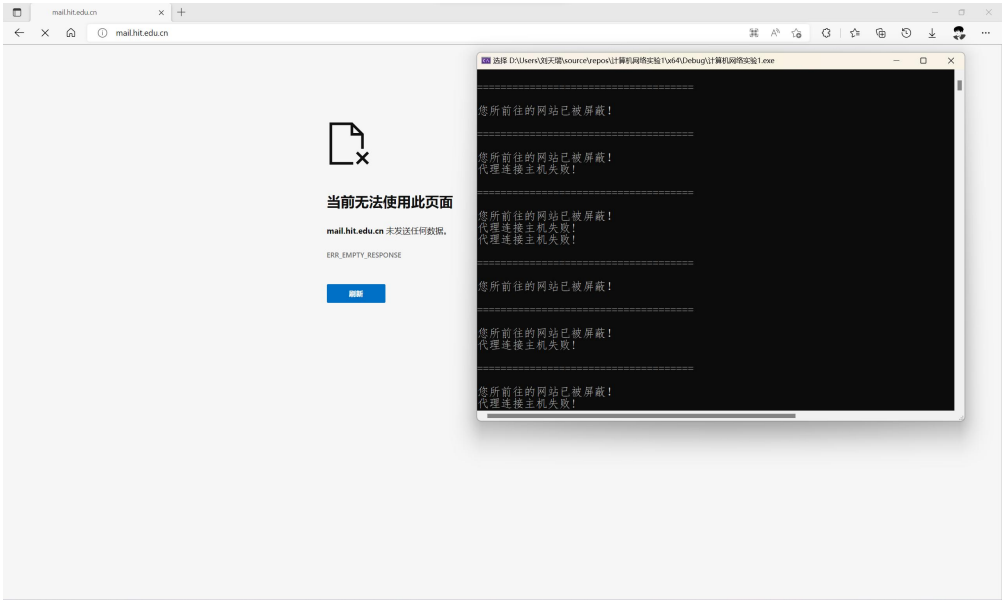



图11 哈工大电子邮件系统网站已经被屏蔽

### 2) 用户过滤

修改程序代码，将允许访问的用户IP地址：127.0.0.1修改为不允许，如图所示：

```
330     if (strcmp(ip, "127.0.0.1") == 1) {
```

图12 修改源代码

重新生成程序进行运行：

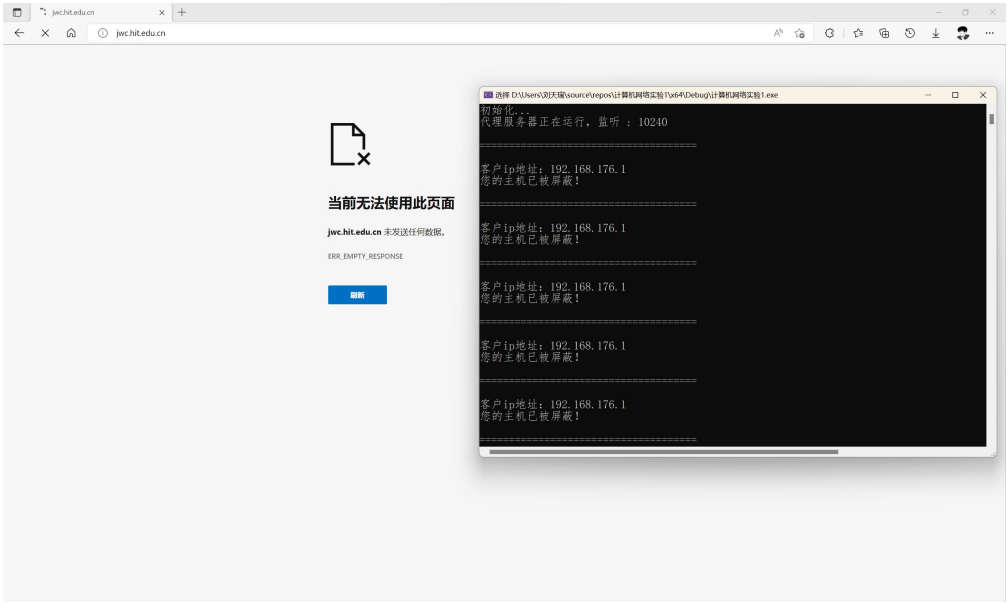



图13 主机用户已经被屏蔽

可以看出代理服务器不会对我们的访问做出反应，用户被成功屏蔽。

**3) 网站引导**

设定的钓鱼源网站为http://today.hit.edu.cn/，目的网站为http://jwts.hit.edu.cn/，在地址栏输入http://today.hit.edu.cn/尝试进行访问：

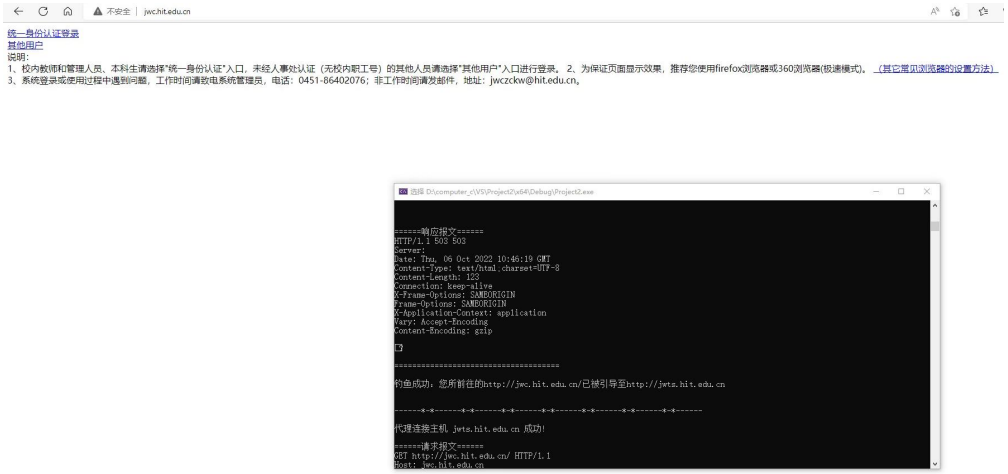



图14 钓鱼源网站已经被成功钓鱼至目的网站

可以看出网页已经被成功重定向，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。

答：

1. 我对于计算机网络实验指导书中的参考代码其中的部分报错的一些思考：

i）其中第一个报错的就是#include "stdafx.h"这第一行代码，我的解决方法是直接去掉或者改为注释即可。我认为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格式的文件都包含相同的include文件，为每个.cpp格式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。

ii）报错的第二个点是代码中使用goto error时，而且在已经调用了goto语句之后还定义了随机变量，大部分的正经工程编译器都会认为这种行为是十分危险的，在visual studio 2022中可以通过对其进行如下的简单设置去除这种错误：我经过查阅资料可知，其余一些编译器需要将所有随机变量都定义在调用goto error之前即可。而且事实上这样做是真正安全的，其他处理方式只是强制忽略这种错误检测，导致存在部分隐患。

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、加载出现错乱或者加载很慢的情况，我猜测是因为对于代理服务器，HTTP发送报文的时候不采用流水线形式发送，因此中间需要多次发送和接受的过程，除此之外，过程中还通过代理服务器这一中介，使得速度变得更慢，因此现在的HTTP报文传送中我猜测其大多使用流水线形式或其他我尚未得知的形式来加速发送报文。

心得体会：

结合实验过程和结果给出实验的体会和收获。

答：

1. 对TCP协议传输数据的流程和方式有了更深的体会；
2. 对socket编程的基本函数和基本操作方法有了初步的了解，通过动手实践有了很大收获，并且产生了浓厚的兴趣；
3. 了解了HTTP代理服务器的基本原理，掌握了HTTP代理服务器设计与编程实现的基本技能，对HTTP请求和响应原理有了为更深刻的认识；
4. 对拓展HTTP代理服务器：网站钓鱼、网站屏蔽、用户屏蔽的简单实现机制有了深刻的认识和理解；
5. 对HTTP Cache缓存的作用有了直观印象。

实验源代码：

```
//#include "stdafx.h"  创建 vs 项目包含的预编译头文件
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <windows.h>
#include <process.h>
#include <string.h>
#include <iostream>
#include <fstream>
#include <Winsock2.h>
```

```
#include <tchar.h>

//使用命名空间
using namespace std;

#pragma comment(lib,"Ws2_32.lib")
//发送数据报文的最大长度
#define MAXSIZE 65507
//http 服务器端口
#define HTTP_PORT 80

//定义 HTTP 重要头部数据文件
struct HttpHeader {
    char method[4];//POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
    char url[1024]; //请求的 url
    char host[1024];//目标主机
    char cookie[1024 * 10];
    HttpHeader(){
        ZeroMemory(this, sizeof(HttpHeader));
    }
};

bool InitSocket();
void ParseHttpHead(char* buffer, HttpHeader* httpHeader);
boolean ParseDate(char* buffer, char* field, char* tempDate);
void makeNewHTTP(char* buffer, char* value);
void makeFilename(char* url, char* filename);
void makeCache(char* buffer, char* url);
void getCache(char* buffer, char* filename);
bool ConnectToServer(SOCKET* serverSocket, char* host);
unsigned int _stdcall ProxyThread(LPVOID lpParameter);
//代理服务器参数
SOCKET ProxyServer;
SOCKADDR_IN ProxyServerAddr;
const int ProxyPort = 10240;
bool haveCache = false;
bool needCache = true;

struct ProxyParam {
    SOCKET cilentSocket;
    SOCKET serverSocket;
};

int t_main(int argc, TCHAR* argv[]) {
```

```
    printf("代理服务器正在启动\n");
    printf("初始化...\n");
    if (!InitSocket()) {
        printf("服务器初始化失败\n");
        return -1;
    }
    printf("代理服务器正在运行，监听 : %d\n", ProxyPort);
    SOCKET acceptSocket = INVALID_SOCKET;
    ProxyParam* lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;
    //代理服务器不断循环监听
    while (true) {
        haveCache = false;
        needCache = true;
        acceptSocket = accept(ProxyServer, NULL, NULL);
        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL) {
            continue;
        }
        //受限用户,与列表中匹配上的都无法访问
        lpProxyParam->cilentSocket = acceptSocket;
        hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0,
0);//注意比较之前将网络二进制的数字转换成网络地址
        CloseHandle(hThread);
        Sleep(200);
    }
    closesocket(ProxyServer);
    WSACleanup();
    return 0;
}

//初始化套接字
bool InitSocket() {
    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    int err; //套接字加载时错误提示
    wVersionRequested = MAKEWORD(2, 2); //版本是 2.2
    //加载 dll 文件 Socket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0) {//找不到 winsock.dll
        printf("加载 winsock.dll 失败，错误：%d", WSAGetLastError());
        return false;
```

```
    }
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {//if 中的语句主要用于比对是否是 2.2 版本
        printf("不能找到正确的winsock版本\n");
WSACleanup();
        return false;
    }
    ProxyServer = socket(AF_INET, SOCK_STREAM, 0); //创建的 TCP 连接的套接字 socket 文件描述符基于 IPV4
    if (INVALID_SOCKET == ProxyServer) {
        printf("创建套接字失败，错误代码为 %d\n", WSAGetLastError());
        return false;
    }
    ProxyServerAddr.sin_family = AF_INET;
    ProxyServerAddr.sin_port = htons(ProxyPort); //指向代理服务器的端口，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
    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
    if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        printf("绑定套接字失败\n");
        return false;
    }
    if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR) {
        printf("监听端口%d 失败", ProxyPort);
        return false;
    }
    return true;
}

//解析 TCP 报文中的 HTTP 头部
void ParseHttpHead(char* buffer, HttpHeader* httpHeader) {
    char* p;
    char* ptr;
    const char* delim = "\r\n";
    p = strtok_s(buffer, delim, &ptr);//提取第一行
    //printf("%s\n", p);
    if (p[0] == 'G') {  //GET 方式
        memcpy(httpHeader->method, "GET", 3);
        memcpy(httpHeader->url, &p[4], strlen(p) - 13);  //url 的长度
    }
    else if (p[0] == 'P') {  //POST 方式
        memcpy(httpHeader->method, "POST", 4);
        memcpy(httpHeader->url, &p[5], strlen(p) - 14);
```

```
    }
    //printf("%s\n", httpHeader->url);
    p = strtok_s(NULL, delim, &ptr);
    while (p) {
        switch (p[0]) {
        case 'H':  //Host
            memcpy(httpHeader->host, &p[6], strlen(p) - 6);
            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;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
}

//分析 HTTP 头部的 field 字段，如果包含该 field 则返回 true，并获取日期
boolean ParseDate(char* buffer, char* field, char* tempDate) {
    char* p, * ptr, temp[5];
    //const char *field = "If-Modified-Since";
    const char* delim = "\r\n";
    ZeroMemory(temp, 5);
    p = strtok_s(buffer, delim, &ptr);
    //printf("%s\n", p);
    int len = strlen(field) + 2;
    while (p) {
        if (strstr(p, field) != NULL) {
            memcpy(tempDate, &p[len], strlen(p) - len);
            //printf("tempDate: %s\n", tempDate);
            return true;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
    return false;
}
```

```
//改造 HTTP 请求报文
void makeNewHTTP(char* buffer, char* value) {
    const char* field = "Host";
    const char* newfield = "If-Modified-Since: ";
    //const char *delim = "\r\n";
    char temp[MAXSIZE];
    ZeroMemory(temp, MAXSIZE);
    char* pos = strstr(buffer, field);
    for (int i = 0; i < strlen(pos); i++) {
        temp[i] = pos[i];
    }
    *pos = '\0';
    while (*newfield != '\0') {  //插入 If-Modified-Since 字段
        *pos++ = *newfield++;
    }
    while (*value != '\0') {
        *pos++ = *value++;
    }
    *pos++ = '\r';
    *pos++ = '\n';
    for (int i = 0; i < strlen(temp); i++) {
        *pos++ = temp[i];
    }
    //printf("buffer: %s\n", buffer);
}

//根据 url 构造文件名
void makeFilename(char* url, char* filename) {
    //char filename[100];  // 构造文件名
    //ZeroMemory(filename, 100);
    char* p = filename;
    while (*url != '\0') {
        if (*url != '/' && *url != ':' && *url != '.') {
            *p++ = *url;
        }
        url++;
    }
}

//进行缓存
void makeCache(char* buffer, char* url) {
    char* p, * ptr, num[10], tempBuffer[MAXSIZE + 1];
    const char* delim = "\r\n";
```

```
    ZeroMemory(num, 10);
    ZeroMemory(tempBuffer, MAXSIZE + 1);
    memcpy(tempBuffer, buffer, strlen(buffer));
    p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
    memcpy(num, &p[9], 3);
    if (strcmp(num, "200") == 0) {  //状态码是 200 时缓存
        //printf("url : %s\n", url);
        char filename[100] = { 0 };  // 构造文件名
        makeFilename(url, filename);
        //printf("filename : %s\n", filename);
        FILE* out;
        if (fopen_s(&out, filename, "wb") == 0) {
            fwrite(buffer, sizeof(char), strlen(buffer), out);
            fclose(out);
        }
        printf("\n 报文已缓存！\n");
    }
}

//获取缓存
void getCache(char* buffer, char* filename) {
    char* p, * ptr, num[10], tempBuffer[MAXSIZE + 1];
    const char* delim = "\r\n";
    ZeroMemory(num, 10);
    ZeroMemory(tempBuffer, MAXSIZE + 1);
    memcpy(tempBuffer, buffer, strlen(buffer));
    p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
    memcpy(num, &p[9], 3);
    if (strcmp(num, "304") == 0) {  //主机返回的报文中的状态码为 304 时返回已缓存的内容
        printf("获取本地缓存！\n");
        ZeroMemory(buffer, strlen(buffer));
        FILE* in;
        if (fopen_s(&in, filename, "rb") == 0) {
            fread(buffer, sizeof(char), MAXSIZE, in);
            fclose(in);
        }
        needCache = false;
    }
}

//根据主机创建目标服务器套接字，并连接
bool ConnectToServer(SOCKET* serverSocket, char* host) {
    SOCKADDR_IN serverAddr;
    serverAddr.sin_family = AF_INET;
```

```
    serverAddr.sin_port = htons(HTTP_PORT); //本地字节顺序 ---> 网络字节顺序
    HOSTENT* hostent = gethostbyname(host);
    if (!hostent) {
        return false;
    }
    IN_ADDR inAddr = *((IN_ADDR*)*hostent->h_addr_list);
    serverAddr.sin_addr.S_un.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
    *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (*serverSocket == INVALID_SOCKET) {
        return false;
    }
    if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) ==
        SOCKET_ERROR) {
        closesocket(*serverSocket);
        return false;
    }
    return true;

}

//线程执行函数
unsigned int _stdcall ProxyThread(LPVOID lpParameter) {
    char Buffer[MAXSIZE], fileBuffer[MAXSIZE];
    char* CacheBuffer, * DateBuffer;
    ZeroMemory(Buffer, MAXSIZE);
    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    //接受客户端的 http 请求
    recvSize = recv(((ProxyParam*)lpParameter)->cilentSocket, Buffer, MAXSIZE, 0);
    HttpHeader* httpHeader = new HttpHeader();
    //if (recvSize <= 0) {
    //  goto error;
    //}
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);
    ParseHttpHead(CacheBuffer, httpHeader);

    //缓存
    DateBuffer = new char[recvSize + 1];
    ZeroMemory(DateBuffer, strlen(Buffer) + 1);
```

```
memcpy(DateBuffer, Buffer, strlen(Buffer) + 1);
//printf("DateBuffer: \n%s\n", DateBuffer);
char filename[100];
ZeroMemory(filename, 100);
makeFilename(httpHeader->url, filename);
//printf("filename : %s\n", filename);
char field[] = "Date";
char date_str[30];  //保存字段 Date 的值
ZeroMemory(date_str, 30);
ZeroMemory(fileBuffer, MAXSIZE);
FILE* in;

//用户过滤
char hostname[128];
int retnew = gethostname(hostname, sizeof(hostname));
HOSTENT *hent = gethostbyname(hostname);
char *ip = inet_ntoa(*(in_addr*)*hent->h_addr_list);  //获取本地 ip 地址
if (strcmp(ip, "127.0.0.1") == 0) {
    printf("\n====================================\n\n");
    printf("客户 ip 地址：%s\n", ip);
    printf("您的主机已被屏蔽！\n");
    goto error;
}

if (fopen_s(&in, filename, "rb") == 0) {
    printf("\n 代理服务器在该 url 下有相应缓存！\n");
    fread(fileBuffer, sizeof(char), MAXSIZE, in);
    fclose(in);
    //printf("fileBuffer : \n%s\n", fileBuffer);
    ParseDate(fileBuffer, field, date_str);
    printf("date_str: %s\n", date_str);
    makeNewHTTP(Buffer, date_str);
    printf("\n======改造后的请求报文======\n%s\n", Buffer);
    haveCache = true;
    goto success;
}

//网站屏蔽
if (strcmp(httpHeader->url, "http://mail.hit.edu.cn/") == 0) {
    printf("\n====================================\n\n");
    printf("您所前往的网站已被屏蔽！\n");
    goto error;
}
```

```
    //网站引导：钓鱼网站
    if (strcmp(httpHeader->url, "http://today.hit.edu.cn/") == 0) {
        printf("\n====================================\n\n");
        printf("钓鱼成功：您所前往的 http://today.hit.edu.cn/已被引导至
http://jwts.hit.edu.cn\n");
        memcpy(httpHeader->host, "jwts.hit.edu.cn", 22);
    }

    delete CacheBuffer;
    delete DateBuffer;

success://成功处理
    if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host)) {
        printf("代理连接主机失败!\n");
        goto error;
    }
    printf("\n\n------*-*------*-*------*-*------*-*------*-*------*-*------*-*------\n
\n");
    printf("代理连接主机 %s 成功!\n", httpHeader->host);
    printf("\n======请求报文======\n%s\n", Buffer);
    //将客户端发送的 HTTP 数据报文直接转发给目标服务器
    ret = send(((ProxyParam*)lpParameter)->serverSocket, Buffer, strlen(Buffer) + 1, 0);
    //等待目标服务器返回数据
    recvSize = recv(((ProxyParam*)lpParameter)->serverSocket, Buffer, MAXSIZE, 0);
    if (recvSize <= 0) {
        printf("接收数据报文失败!\n");
        goto error;
    }
    //有缓存时，判断返回的状态码是否是 304，若是则将缓存的内容发送给客户端
    if (haveCache == true) {
        getCache(Buffer, filename);
    }
    //将目标服务器返回的数据直接转发给客户端
    printf("\n======响应报文======\n%s\n", Buffer);
    if (needCache == true) {
        makeCache(Buffer, httpHeader->url);  //缓存报文
    }
    ret = send(((ProxyParam*)lpParameter)->cilentSocket, Buffer, sizeof(Buffer), 0);

error://错误处理
    //printf("关闭套接字\n");
    Sleep(200);
    closesocket(((ProxyParam*)lpParameter)->cilentSocket);
    closesocket(((ProxyParam*)lpParameter)->serverSocket);
```

```
    delete lpParameter;
    _endthreadex(0);
    return 0;
}
```