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	张智雄		院系	计算学部	
班级	2103601		学号	2021112845	
任课教师	刘亚维		指导教师	刘亚维	
实验地点	格物 207		实验时间	2023.10.21	
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					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

实验目的：

1. 熟悉并掌握 Socket 网络编程的过程与技术；
2. 深入理解 HTTP 协议，掌握 HTTP 代理服务器的基本工作原理；
3. 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：

一、实验总体思路

首先了解一下客户端和服务器端的基本任务：

客户端（Client）	服务器端（Server）
a) 根据目标服务器IP地址与端口号创建套接字，并连接服务器； b) 发送请求报文； c) 接收返回报文； d) 关闭连接；	a) 对到来的请求创建套接字，绑定套接字的IP地址和端口号，对端口进行监听； b) 等待入连接请求； c) 从套接字中读取请求； d) 对请求进行响应，发送响应数据； e) 关闭连接；

本实验实现的即是一个HTTP代理服务器，接收并发送来自客户的HTTP请求，同时转发来自HTTP服务器的响应报文到客户端。在此过程中，既充当客户端，又充当服务器端的角色。

二、实验基础代理部分

1.主函数

总体上使用InitSocket()函数初始化套接字socket，利用while(true)循环与listen函数实现对指定端口的持续监听；使用accept函数接收请求，同时创建子线程进行报文的转发响应；处理完成后，等待200ms关闭该线程，并清理缓存；重复循环处理下一个请求。

```
int main(int argc, char* argv[])
{
    printf("代理服务器正在启动\n");
    printf("初始化...\n");
    if (!InitSocket()) {
        printf("socket 初始化失败\n");
        return -1;
    }
    printf("代理服务器正在运行，监听端口 %d\n", ProxyPort);

    SOCKET acceptSocket = INVALID_SOCKET; // 初始化接收套接字
    ProxyParam* lpProxyParam; //初始化代理参数，内包含客户端和服务端套接字
    HANDLE hThread;
    DWORD dwThreadID;
```

```
    //代理服务器不断监听
    while (true) {
        acceptSocket = accept(ProxyServer, NULL, NULL);
        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL) {
            continue;
        }
        lpProxyParam->clientSocket = acceptSocket;

        // 创建子线程
        hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0);

        CloseHandle(hThread);
        Sleep(200);
    }
    closesocket(ProxyServer);
    WSACleanup();
    return 0;
}
```

图 1 主函数代码截图

2. InitSocket()函数初始化套接字

此函数主要分为两步：加载套接字库和初始化套接字。

i). 加载套接字库。此步骤加载Socket库，并检查winsock.dll的加载是否成功以及版本是否匹配。

```
//加载套接字库（必须）
WORD wVersionRequested;
WSADATA wsaData;
//套接字加载时错误提示
int err;
//两个byte型合成一个16位字，表示Winsock库版本2.2
wVersionRequested = MAKEWORD(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 失败，错误代码为: %d\n", WSAGetLastError());
    return FALSE;
}

// 获得低位字节和高位字节，判断版本是否匹配
if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
{
    printf("不能找到正确的winsock 版本\n");
    WSACleanup();
    return FALSE;
}
```

ii). 初始化套接字。利用socket(AF_INET,SOCK_STREAM)方法创建套接字，第一个参数代表协议族，AF_INET表示是IPV4地址簇；第二个参数代表套接字类型，SOCK_STREAM表示是面向TCP连接的流式套接字；有时后面还会有第三个参数，代表协议号，默认设置为0；而后使用bind()方法将套接字与本机地址及响应端口绑定，并设置为监听状态。

```
//AF_INET IPV4地址簇; SOCK_STREAM TCP连接，SOCK_DGRAM UDP连接
ProxyServer = socket(AF_INET, SOCK_STREAM, 0);
if (ProxyServer == INVALID_SOCKET) {
    printf("创建套接字失败，错误代码为: %d\n", WSAGetLastError());
    return FALSE;
}

ProxyServerAddr.sin_family = AF_INET; // IPv4
ProxyServerAddr.sin_port = htons(ProxyPort); //指定端口
```

```
//ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY; //监听所有可用网络接口上的连接请求
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //仅本机用户可访问服务器

//绑定套接字
if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
    printf("绑定套接字失败\n");
    return FALSE;
}
//设置套接字为监听状态
if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR) {
    printf("监听端口%d 失败", ProxyPort);
    return FALSE;
}
return TRUE;
```

图 2 InitSocket()函数代码截图

3. ProxyThread()线程处理函数

在线程函数中，首先需要使用ZeroMemory()方法初始化内存，再使用recv()函数接收来自客户端的HTTP请求，消息内容缓存在Buffer中，recvSize为实际收到的报文字节数，而后使用ParseHttpHead函数对HTTP报文首部进行解析。

```
unsigned int __stdcall ProxyThread(LPVOID lpParameter) {

    char Buffer[MAXSIZE];
    char* CacheBuffer;
    ZeroMemory(Buffer, MAXSIZE);
    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;

    BOOL haveCache = false;
    BOOL needCache = true;
    char fileBuffer[MAXSIZE];
    char filename[100];
    ZeroMemory(filename, 100);

    HttpHeader* httpHeader = new HttpHeader();
    //接受来自客户端的报文
    recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
    if (recvSize <= 0) {
        goto error;
    }
    //printf("%s\n", Buffer);
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);

    //解析http首部
    if (!ParseHttpHead(CacheBuffer, httpHeader))
    {
        goto error;
    }
    delete[] CacheBuffer;
```

之后调用ConnectToServer函数，根据发送端套接字的协议族和端口号还有套接字类型，以及目的主机的IP地址和端口号进行建立和服务器之间的连接。连接成功后，调用send()将客户端发送的HTTP请求报文转发给目标服务器。

```
// 是否连接到需要访问的网址
if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host)) {
    goto error;
}

//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);
```

接下来调用recv()函数等待目标服务器返回数据，可以理解为网页内容， 接受之后将返回的数据直接转发给客户端，结束本次线程处理。

```
//等待目标服务器返回数据
recvSize = recv(((ProxyParam*)lpParameter)->serverSocket, Buffer, MAXSIZE, 0);
if (recvSize <= 0) {
    goto error;
}

//将目标服务器返回的数据直接转发给客户端
ret = send(((ProxyParam*)lpParameter)->clientSocket, Buffer, sizeof(Buffer), 0);
```

最后是异常处理，如果在过程中有异常均跳转到error，结束线程运行。

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

图 3 ProxyThread()函数代码截图

4. ParseHttpHead()HTTP头部解析函数

根据下图HTTP请求报文头部结构，使用strtok_s()方法对报文信息进行分割提取，得到方法、URL、Host、Cookie等信息。

图 4 HTTP 请求报文头结构

```
p = strtok_s(buffer, delim, &ptr);

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

// 提取后续行
p = strtok_s(NULL, delim, &ptr);
```

```
// 提取后续行
p = strtok_s(NULL, delim, &ptr);
while (p) {
    switch (p[0]) {
        case 'H'://Host
            memcpy(httpHeader->host, &p[6], strlen(p) - 6);
            if (strcmp(httpHeader->method,"POST") == 0 || strcmp(httpHeader->method, "GET") == 0)
                printf("Host:%s\n", httpHeader->host);
            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);
                }
                if (strcmp(httpHeader->method, "POST") == 0 || strcmp(httpHeader->method, "GET") == 0)
                    printf("Cookie:%s\n", httpHeader->cookie);
            }
            break;

        default:
            break;
    }
    p = strtok_s(NULL, delim, &ptr);
}
if (httpHeader->method == "POST" || httpHeader->method == "GET")
    printf("=====================================\n");
return true;
}
```

图 5 ParseHttpHead()函数代码截图

5. ConnectToServer ()函数连接服务器

与InitSocket()函数中创建套接字过程类似，根据发送端套接字的协议簇和端口号还有套接字类型，以及目的主机的IP地址和端口号进行建立连接，如果连接成功，放回TRUE。

```
BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(HTTP_PORT);
    HOSTENT* hostent = gethostbyname(host);
    if (!hostent) {
        return FALSE;
    }
    in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
    serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
    // 创建套接字
    *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;
}
```

图 6 ConnectToServer()代码截图

三、Cache部分功能设计

当访问某网站时，首先通过 URL 对应的文件名寻找本地对应缓存文件，

1) 若无对应文件，即为第一次访问某网站，则代理服务器通过 writeinCache()函数将该请求返回的响应数据写入缓存即相应文件中

2) 若匹配到本地缓存文件，则获取文件中的 Date 信息，利用 MakeNewHTTP()函数构造条件 GET 报文，即为报文在 Host 后插入 If-Modified-Since 头部行。

```
if ((f = fopen(filename, "rb")) != NULL)
{
    fread(fileBuffer, sizeof(char), MAXSIZE, f);
    fclose(f);
    readDate(fileBuffer, field, date_str);
    printf("date_str:%s\n", date_str);
    makeNewHTTP(Buffer, date_str);
    haveCache = true;
}
```

图 7 ProxyThread()函数中构造条件 GET 代码截图

再向服务器端发送请求，通过服务器返回的数据码判断是否为最新的数据，若返回 304，则内容并未再次更新，直接使用 readCache()方法读取缓存中的内容并转发给客户端；若返回 200，则将此响应报文直接发给客户端，同时更新本地缓存。

由于上述 writeinCache()，readCache()，MakeNewHTTP()均为简单的文件读写、字符串插入等操作，此处不再给出截图。

四、扩展功能设计

a) 网站过滤：允许/不允许访问某些网站；

设置字符串数组存储屏蔽网站地址，对请求的 HTTP 报文头部进行解析，提取其中的访问地址 URL，并与屏蔽网站地址进行匹配，若匹配成功，则代码跳转至 error 部分，打印相关提示信息，立即关闭套接字，断开连接。

```
//屏蔽网站功能：
if (strcmp(httpHeader->url, INVILID_WEBSITE) == 0)
{
    printf("%s网站已被屏蔽\n", httpHeader->url);
    goto error;
}
```

b) 用户过滤：支持/不支持某些用户访问外部网站；

一种方法是直接更改套接字绑定的主机地址，绑定主机 127.0.0.1 即限制仅本机用户可访问服务器 ProxyServerAddr.sin_addr.S_un.S_addr=inet_addr("127.0.0.1")。

也可以在 accept()监听套接字时获取客户端 IP 与禁止访问 IP 进行字符串比较以实现用户屏蔽。

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

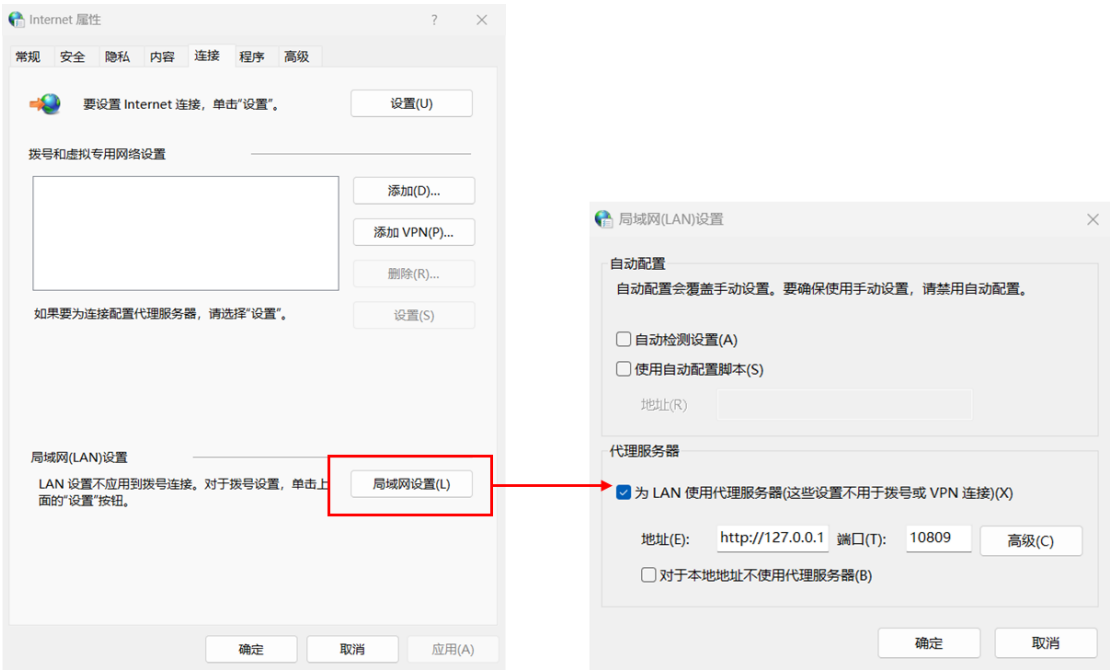
设置字符串数组存储钓鱼网站地址，设置引导目的网站地址，同样解析匹配 URL，若匹配成功，则更改 HTTP 头部字段的访问网址 URL 与 Host 主机地址，实现网页的钓鱼跳转。

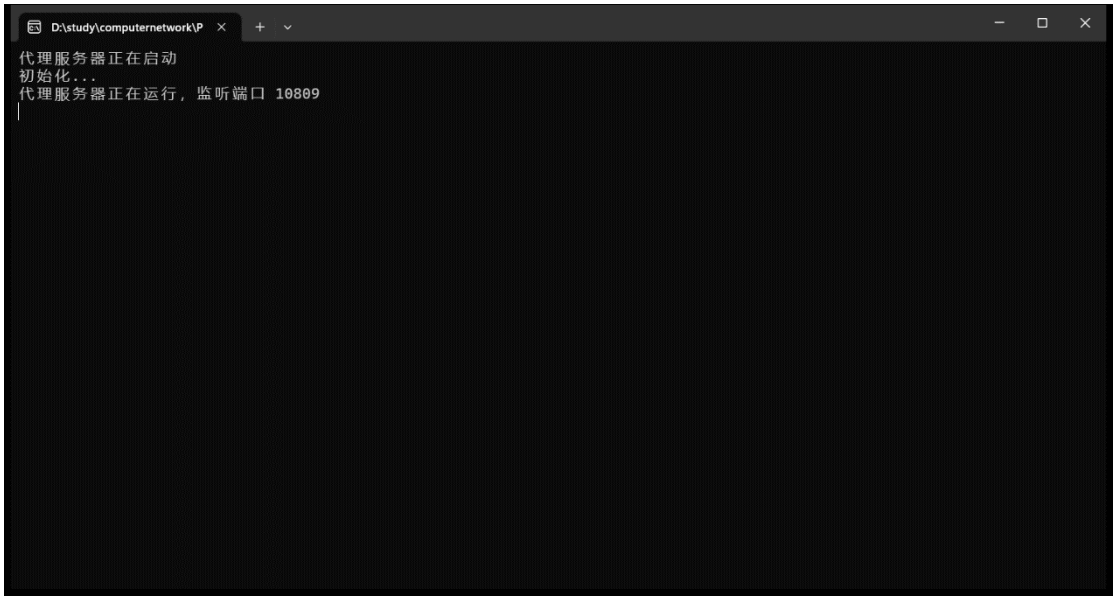
```
//网站引导：将访问网址转到其他网站
if (strcmp(httpHeader->url, FISH_WEB_SRC) == 0)
{
    printf("%s目标网址已被引导至%s\n", httpHeader->url, FISH_WEB_SRC);
    memcpy(httpHeader->host, fish_web_host, strlen(fish_web_host) + 1);
    memcpy(httpHeader->url, fish_web_url, strlen(fish_web_url));
}
```

实验结果：

1. 浏览器配置代理，运行程序

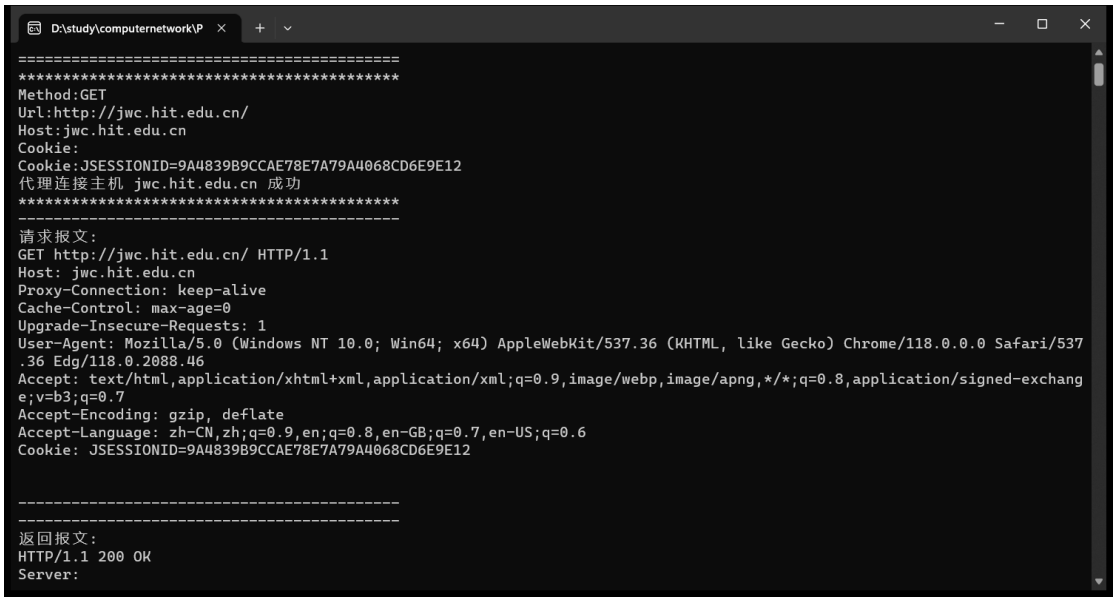
打开代理，设置好 IP 地址和端口号，运行程序监听端口 10809，这时候正常 Https 协议的网站是无法访问的。





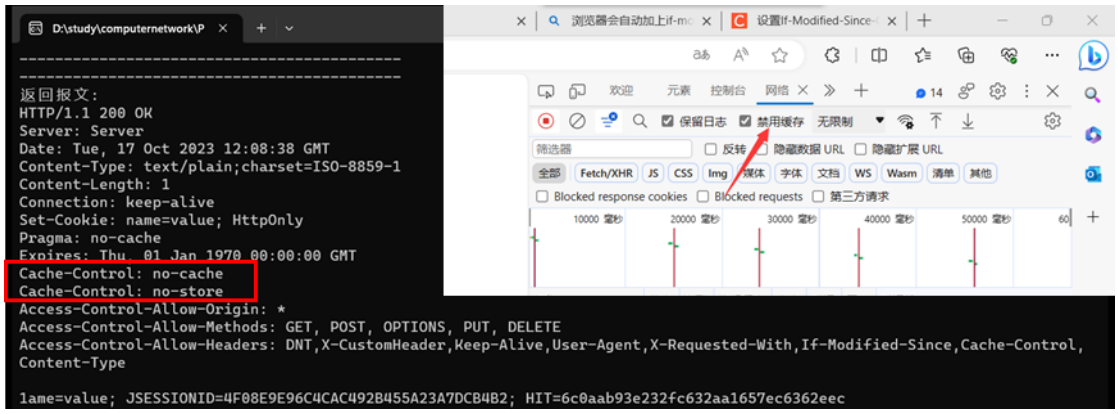
2. 实现一个基本 HTTP 代理服务器

访问哈工大教务处：



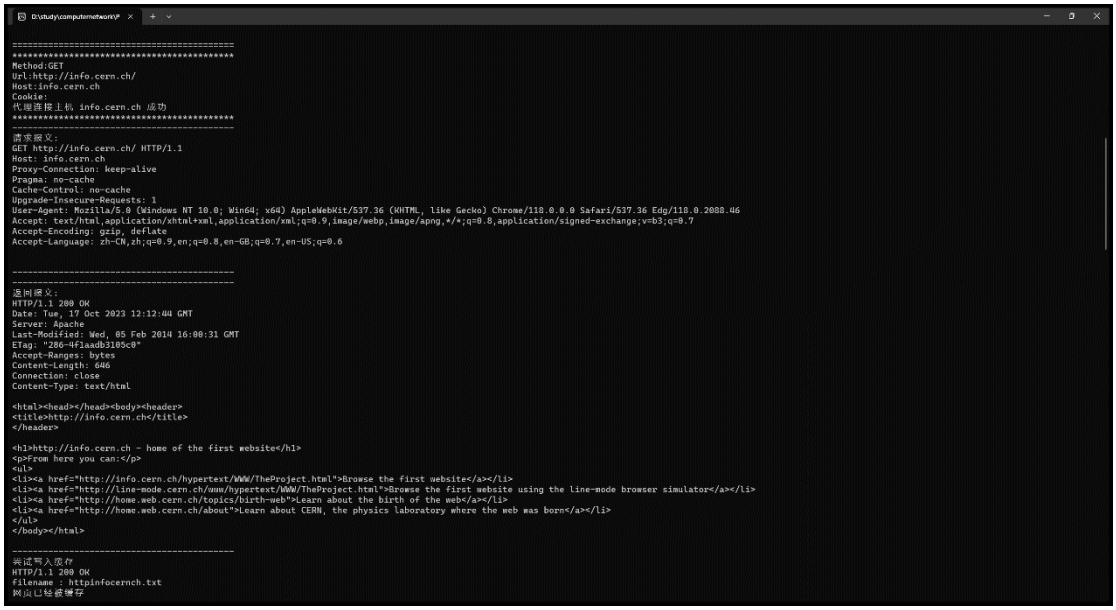
3. 实现Cache功能的HTTP代理服务器

观察返回报文发现 jwts.hit.edu.cn 等网站不支持缓存（接受条件 GET 始终返回 200），因而改用其他网站，同时关闭浏览器自带的缓存功能。

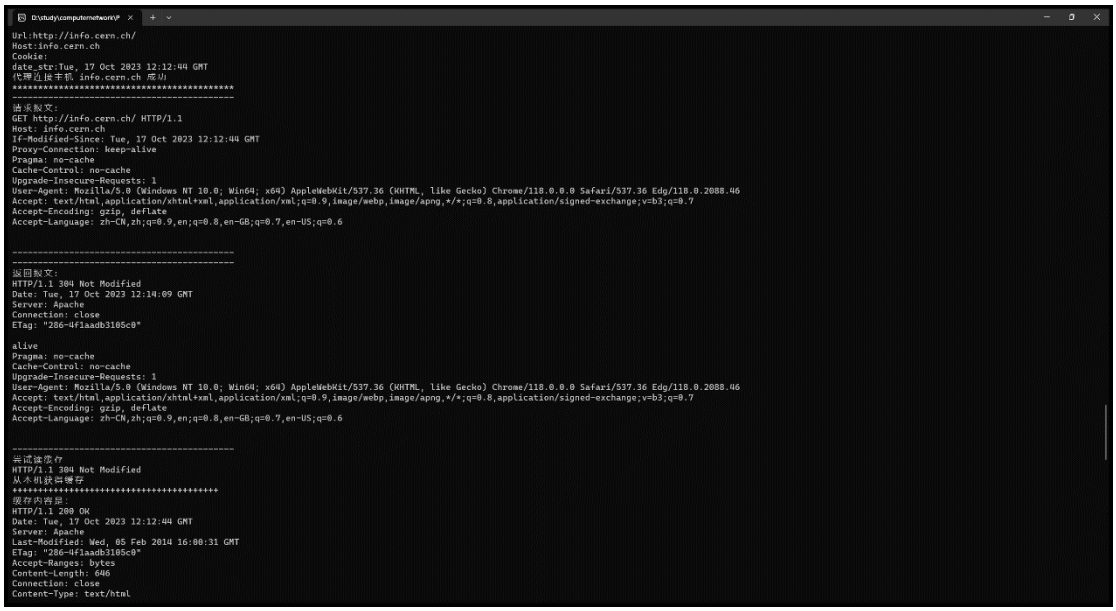


在网站 http://info.cern.ch 进行缓存读写的测试：

a) 首次访问时：

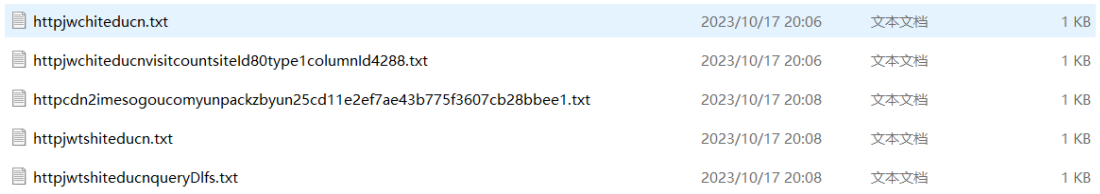


b) 再次访问时：

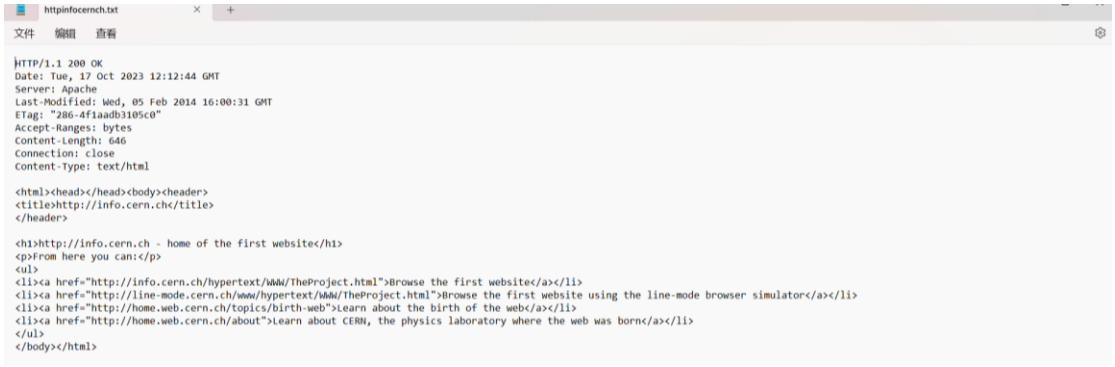


c) 针对条件 GET 返回 200 的情况，程序会重新写入本地文件，即重新缓存：

缓存文件列表（不完全）：



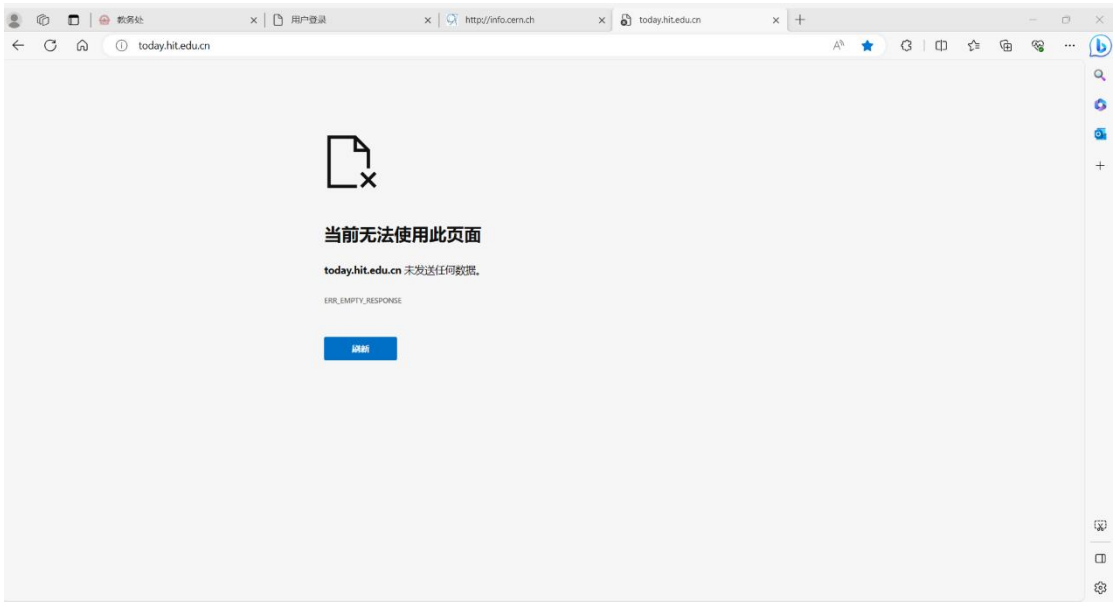
观察缓存内容，包含 Last-Modified 等信息：

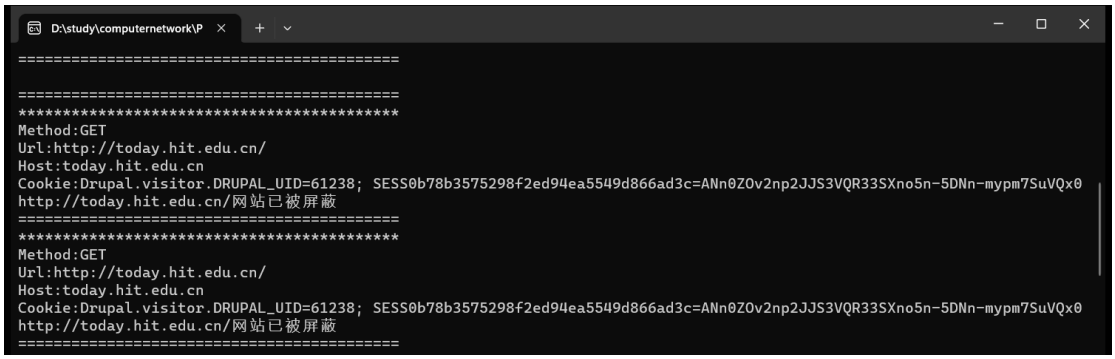


4. 实现拓展功能的HTTP代理服务器

a) 网站过滤：允许/不允许访问某些网站：

设置屏蔽今日哈工大网站 http://today.hit.edu.cn/，结果如下：



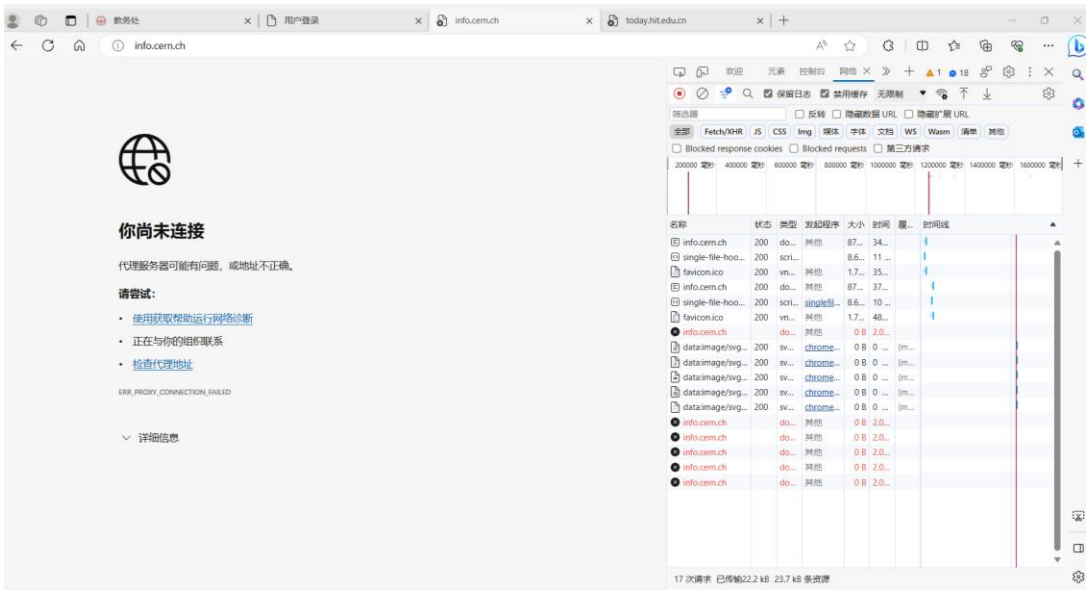


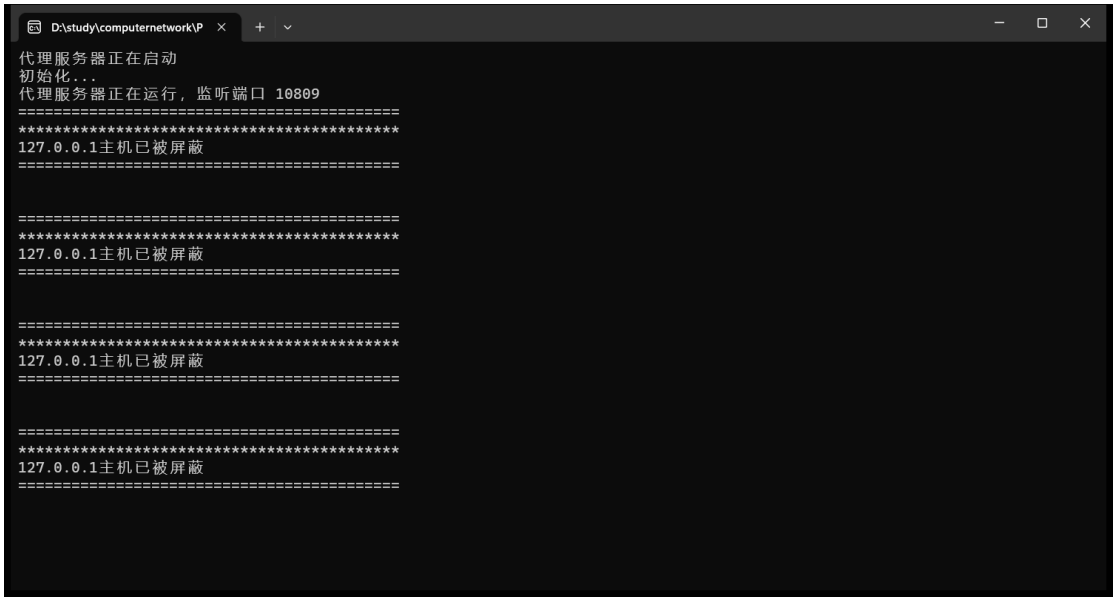
b) 用户过滤：支持/不支持某些用户访问外部网站：

更改套接字绑定的主机地址，绑定主机 127.0.0.1 即限制仅本机用户可访问服务器 ProxyServerAddr.sin_addr.S_un.S_addr=inet_addr("127.0.0.1")。

也可以在 accept()监听套接字时获取客户端 IP 与禁止访问 IP 进行字符串比较以实现用户屏蔽。

以下将屏蔽主机设为 127.0.0.1，本机即不能正常访问服务器：

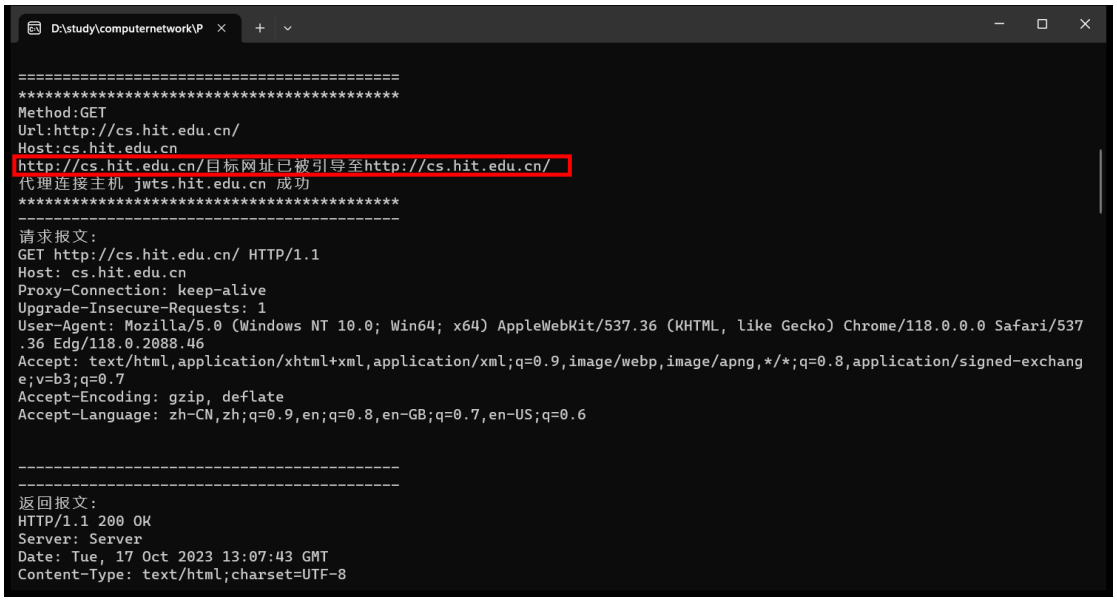




c) 网站引导：将用户对某个网站的访问引导至一个模拟网站

设置字符串数组存储钓鱼网站地址，设置引导目的网站地址，同样解析匹配 URL，若匹配成功，则更改 HTTP 头部字段的访问网址 URL 与 Host 主机地址，实现网页的钓鱼跳转。

以下将网站 http://cs.hit.edu.cn/引导到网站 http://jwts.hit.edu.cn/，没有图像的原因应该是没有拦截后续请求图片的报文：



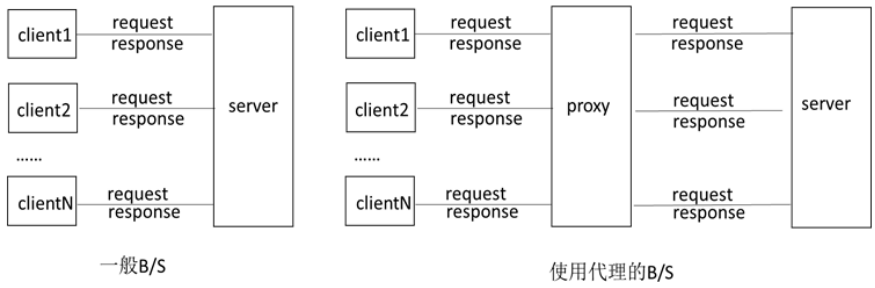
问题讨论：

1. Socket编程的客户端和服务端主要步骤

客户端（Client）	服务器端（Server）
a) 根据目标服务器IP地址与端口号创建套接字，并连接服务器； b) 发送请求报文； c) 接收返回报文； d) 关闭连接；	a) 对到来的请求创建套接字，绑定套接字的IP地址和端口号，对端口进行监听； b) 等待入连接请求； c) 从套接字中读取请求； d) 对请求进行响应，发送响应数据； e) 关闭连接；

2. HTTP代理服务器原理

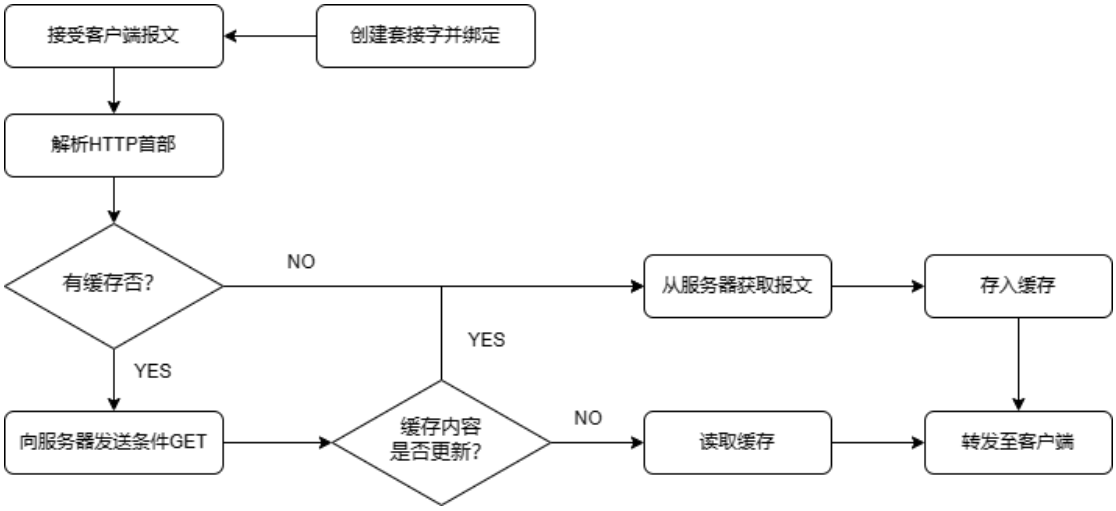
代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。如下图所示，为普通Web应用通信方式与采用代理服务器的通信方式的对比。



代理服务器的工作原理如下：

1) 代理服务器在指定端口监听浏览器的访问请求，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，找到对象文件后，提取该对象文件的最新被修改时间；
2) 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。
3) 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。

3. HTTP代理服务器流程图



4. 实现HTTP代理服务器的关键技术及解决方案

a) 单用户代理服务器

单用户的简单代理服务器可以设计为一个非并发的循环服务器。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，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；创建访问原（目标）服务器的TCP套接字，将HTTP请求报文转发给目标服务器，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，最后关闭套接字，等待下一次连接。

b) 多用户代理服务器

多用户的简单代理服务器可以实现为一个多线程并发服务器。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，服务结束之后子线程终止。与此同时，主线程继续接受下一个客户的代理服务。

5. HTTP代理服务器实验验证过程以及实验结果

见上文“实验结果”部分。

6. HTTP代理服务器源代码

```
#define _CRT_SECURE_NO_WARNINGS
//#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>
#pragma comment(lib,"Ws2_32.lib")
#define MAXSIZE 65507 //发送数据报文的最大长度
#define HTTP_PORT 80 //http 服务器端口

#define INVILID_WEBSITE "http://today.hit.edu.cn/"     //屏蔽网址
#define BAN_CLIENT "127.0.0.2"                                  //屏蔽主机
#define FISH_WEB_SRC "http://cs.hit.edu.cn/"           //钓鱼源网址
#define fish_web_url "http://jwts.hit.edu.cn/"          //钓鱼目的网址
#define fish_web_host "jwts.hit.edu.cn"                    //钓鱼目的地址的主机名


//Http 重要头部数据
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]; //cookie
 HttpHeader() {
        ZeroMemory(this, sizeof(HttpHeader));
 }
};

//代理相关参数
SOCKET ProxyServer;                 // 套接字
sockaddr_in ProxyServerAddr; // 套接字地址结构
const int ProxyPort = 10809; // 代理端口号


struct ProxyParam {
 // 代理参数，分别定义客户端和服务端套接字
 SOCKET clientSocket;
 SOCKET serverSocket;
};

BOOL InitSocket();
```

```
BOOL ParseHttpHead(char* buffer, HttpHeader* httpHeader);
BOOL ConnectToServer(SOCKET* serverSocket, char* host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void writeinCache(char* buffer, char* url);
void readCache(char* buffer, char* url, char* filename);
void makeNewHTTP(char* buffer, char* value);
void readDate(char* buffer, char* field, char* tempDate);
void Filename(char* url, char* filename);


//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
//可以使用线程池技术提高服务器效率
//const int ProxyThreadMaxNum = 20;
//HANDLE ProxyThreadHandle[ProxyThreadMaxNum] = {0};
//DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};

int main(int argc, char* argv[])
{
 printf("代理服务器正在启动\n");
 printf("初始化...\n");
 if (!InitSocket()) {
        printf("socket  初始化失败\n");
        return -1;
 }
 printf("代理服务器正在运行，监听端口  %d\n", ProxyPort);

 SOCKET acceptSocket = INVALID_SOCKET; //  初始化接收套接字
 ProxyParam* lpProxyParam; //初始化代理参数，内包含客户端和服务端套接字
 HANDLE hThread;
 DWORD dwThreadID;

 //代理服务器不断监听
 while (true) {
        sockaddr_in caddr;
        int length = sizeof(caddr);
        acceptSocket = accept(ProxyServer, (sockaddr*)&caddr, &length);
        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL) {
                continue;
        }
        char* ip = inet_ntoa(caddr.sin_addr);
        if (strcmp(ip, BAN_CLIENT) == 0) {
                printf("========================================\n");
                printf("****************************************\n");
```

```
            printf("%s主机已被屏蔽\n", ip);
            printf("================================================\n\n\n");
            continue;
        }

        lpProxyParam->clientSocket = acceptSocket;

        // 创建子线程
        hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0);

        CloseHandle(hThread);
        Sleep(200);
    }
    closesocket(ProxyServer);
    WSACleanup();
    return 0;
}

//************************************
// Method: InitSocket
// FullName: InitSocket
// Access: public
// Returns: BOOL
// Qualifier: 初始化套接字
//************************************
BOOL InitSocket() {

    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //两个byte型合成一个16位字，表示Winsock库版本2.2
    wVersionRequested = MAKEWORD(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 失败，错误代码为: %d\n", WSAGetLastError());
        return FALSE;
    }

    // 获得低位字节和高位字节，判断版本是否匹配
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
```

```
    {
        printf("不能找到正确的winsock 版本\n");
        WSACleanup();
        return FALSE;
    }

    //AF_INET IPV4地址簇; SOCK_STREAM TCP连接，SOCK_DGRAM UDP连接
    ProxyServer = socket(AF_INET, SOCK_STREAM, 0);
    if (ProxyServer == INVALID_SOCKET) {
        printf("创建套接字失败，错误代码为：%d\n", WSAGetLastError());
        return FALSE;
    }

    ProxyServerAddr.sin_family = AF_INET; // IPv4
    ProxyServerAddr.sin_port = htons(ProxyPort); //指定端口
    //ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY; //监听所有可用网络接口上的连接请求
    ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //仅本机用户可访问服务器

    //绑定套接字
    if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
{
        printf("绑定套接字失败\n");
        return FALSE;
    }
    //设置套接字为监听状态
    if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR) {
        printf("监听端口%d 失败", ProxyPort);
        return FALSE;
    }
    return TRUE;
    }


    //***********************************
    // Method: ProxyThread
    // FullName: ProxyThread
    // Access: public
    // Returns: unsigned int __stdcall
    // Qualifier: 线程执行函数
    // Parameter: LPVOID lpParameter
    // 返回无符号整数，__stdcall说明函数从右向左通过堆栈传递
    //***********************************
    unsigned int __stdcall ProxyThread(LPVOID lpParameter) {
```

```
char Buffer[MAXSIZE];
char* CacheBuffer;
ZeroMemory(Buffer, MAXSIZE);
SOCKADDR_IN clientAddr;
int length = sizeof(SOCKADDR_IN);
int recvSize;
int ret;

BOOL haveCache = false;
BOOL needCache = true;
char fileBuffer[MAXSIZE];
char filename[100];
ZeroMemory(filename, 100);

HttpHeader* httpHeader = new HttpHeader();
//接受来自客户端的报文
recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
if (recvSize <= 0) {
	goto error;
}

CacheBuffer = new char[recvSize + 1];
ZeroMemory(CacheBuffer, recvSize + 1);
memcpy(CacheBuffer, Buffer, recvSize);

//解析http首部
if (!ParseHttpHead(CacheBuffer, httpHeader))
{
	goto error;
}
delete[] CacheBuffer;

FILE* f;
Filename(httpHeader->url, filename);
char* field;
field = (char*)"Date";
char date_str[30];
ZeroMemory(date_str, 30);
// 是否已经有缓存
if ((f = fopen(filename, "rb")) != NULL)
{
	fread(fileBuffer, sizeof(char), MAXSIZE, f);
	fclose(f);
	readDate(fileBuffer, field, date_str);
```

```
        printf("date_str:%s\n", date_str);
        makeNewHTTP(Buffer, date_str);
        haveCache = true;
    }

    //屏蔽网站功能：
    if (strcmp(httpHeader->url, INVILID_WEBSITE) == 0)
    {
        printf("%s网站已被屏蔽\n", httpHeader->url);
        goto error;
    }

    //网站引导：将访问网址转到其他网站
    if (strcmp(httpHeader->url, FISH_WEB_SRC) == 0)
    {
        printf("%s目标网址已被引导至%s\n", httpHeader->url, FISH_WEB_SRC);
        memcpy(httpHeader->host, fish_web_host, strlen(fish_web_host) + 1);
        memcpy(httpHeader->url, fish_web_url, strlen(fish_web_url));
    }

    // 是否连接到需要访问的网址
    if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host)) {
        goto error;
    }
    printf("代理连接主机 %s 成功\n", httpHeader->host);
    printf("********************************************\n");

    printf("--------------------------------------------\n");
    printf("请求报文:\n%s\n", Buffer);
    printf("--------------------------------------------\n");
    //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) {
        goto error;
    }

    printf("--------------------------------------------\n");
    printf("返回报文:\n%s\n", Buffer);
    printf("--------------------------------------------\n");
    if (haveCache){
        printf("尝试读缓存\n");
```

```
        readCache(Buffer, httpHeader->url, filename);
    }
    else{
        printf("尝试写入缓存\n");
        writeinCache(Buffer, httpHeader->url);
    }

    //将目标服务器返回的数据直接转发给客户端
    ret = send(((ProxyParam*)lpParameter)->clientSocket, Buffer, sizeof(Buffer), 0);

    //错误处理
error:
    //printf("关闭套接字\n");
    Sleep(200);
    closesocket(((ProxyParam*)lpParameter)->clientSocket);
    closesocket(((ProxyParam*)lpParameter)->serverSocket);
    delete lpParameter;
    _endthreadex(0);
    return 0;
}


//************************************
// Method: ParseHttpHead
// FullName: ParseHttpHead
// Access: public
// Returns: void
// Qualifier:  解析TCP 报文中的HTTP 头部
// Parameter: char * buffer
// Parameter: HttpHeader * httpHeader
//************************************
BOOL ParseHttpHead(char* buffer, HttpHeader* httpHeader) {
    char* p;
    char* ptr;
    const char* delim = "\r\n";

    //提取第一行
    p = strtok_s(buffer, delim, &ptr);

    if (p[0] == 'G') {//GET 方式
        printf("=============================================\n");
        printf("*********************************************\n");
        memcpy(httpHeader->method, "GET", 3);
        memcpy(httpHeader->url, &p[4], strlen(p) - 13);
```

```
            printf("Method:%s\n", httpHeader->method);
            printf("Url:%s\n", httpHeader->url);
        }
        else if (p[0] == 'P') {//POST 方式
            printf("==========================================\n");
            printf("******************************************\n");
            memcpy(httpHeader->method, "POST", 4);
            memcpy(httpHeader->url, &p[5], strlen(p) - 14);
            printf("Method:%s\n", httpHeader->method);
            printf("Url:%s\n", httpHeader->url);
        }

        // 提取后续行
        p = strtok_s(NULL, delim, &ptr);
        while (p) {
            switch (p[0]) {
                case 'H'://Host
                    memcpy(httpHeader->host, &p[6], strlen(p) - 6);
                    if (strcmp(httpHeader->method,"POST") == 0 || strcmp(httpHeader->method, "GET") ==
0)
                        printf("Host:%s\n", httpHeader->host);
                    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);
                        }
                        if (strcmp(httpHeader->method, "POST") == 0 || strcmp(httpHeader->method,
"GET") == 0)
                            printf("Cookie:%s\n", httpHeader->cookie);
                    }
                    break;

                default:
                    break;
            }
            p = strtok_s(NULL, delim, &ptr);
        }
        if (httpHeader->method == "POST" || httpHeader->method == "GET")
            printf("==========================================\n");
```

```
 return true;
}


//***********************************
// Method: ConnectToServer
// FullName: ConnectToServer
// Access: public
// Returns: BOOL
// Qualifier: 根据主机创建目标服务器套接字，并连接
// Parameter: SOCKET * serverSocket
// Parameter: char * host
//***********************************
BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
 sockaddr_in serverAddr;
 serverAddr.sin_family = AF_INET;
 serverAddr.sin_port = htons(HTTP_PORT);
 HOSTENT* hostent = gethostbyname(host);
 if (!hostent) {
        return FALSE;
 }
 in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
 serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
 // 创建套接字
 *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;
}


void writeinCache(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(tempBuffer, delim);//提取第一行
 memcpy(num, &p[9], 3);
 printf("%s\n", p);
 if (strcmp(num, "200") == 0) {
        //状态码是200时缓存
        char filename[100];
        ZeroMemory(filename, 100);
        Filename(url, filename);
        printf("filename : %s\n", filename);
        FILE* f;
        f = fopen(filename, "w");
        fwrite(buffer, sizeof(char), strlen(buffer), f);
        fclose(f);
        printf("网页已经被缓存\n");
 }
 printf("========================================\n\n");
}

void readCache(char* buffer, char* url, 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(tempBuffer, delim);//提取第一行
 memcpy(num, &p[9], 3);
 printf("%s\n",p);
 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("从本机获得缓存\n");
        ZeroMemory(buffer, strlen(buffer));
        FILE* f = NULL;
        if ((f = fopen(filename, "r")) != NULL) {
               fread(buffer, sizeof(char), MAXSIZE, f);
               fclose(f);
        }
        printf("++++++++++++++++++++++++++++++++++++++++\n");
        printf("缓存内容是：\n%s", buffer);
        printf("++++++++++++++++++++++++++++++++++++++++\n");
 }
 else if (strcmp(num, "200") == 0) {
```

```
            //状态码是200时缓存
            char filename[100];
            ZeroMemory(filename, 100);
            Filename(url, filename);
            printf("filename : %s\n", filename);
            FILE* f;
            f = fopen(filename, "w");
            fwrite(buffer, sizeof(char), strlen(buffer), f);
            fclose(f);
            printf("网页已经重新缓存\n");
    }
    printf("========================================\n\n");
}

/*构造条件GET*/
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);

    //在Host后插入If-Modified-Since字段
    char* pos = strstr(buffer, field);
    int i = 0;
    while (*pos != '\n') {
            pos++;
    }
    pos++;
    for (i = 0; i < strlen(pos); i++) {
            temp[i] = pos[i];
    }
    *pos = '\0';
    while (*newfield != '\0') {
            *pos++ = *newfield++;
    }
    while (*value != '\0') {
            *pos++ = *value++;
    }
    *pos++ = '\r';
    *pos++ = '\n';

    for (i = 0; i < strlen(temp); i++) {
            *pos++ = temp[i];
```

```
    }
}

void readDate(char* buffer, char* field, char* tempDate) {
    char* p, * ptr, temp[5];
    ZeroMemory(temp, 5);

    const char* delim = "\r\n";
    p = strtok(buffer, delim); //  按行读取
    int len = strlen(field) + 2;
    while (p) {
        if (strstr(p, field) != NULL) {
            //  如果p中包含field字串，将&p[6]copy给tempdate
            memcpy(tempDate, &p[len], strlen(p) - len);
        }
        p = strtok(NULL, delim);
    }
}

void Filename(char* url, char* filename) {
    int count = 0;
    while (*url != '\0') {
        if ((*url >= 'a' && *url <= 'z') || (*url >= 'A' && *url <= 'Z') || (*url >= '0' && *url <= '9')) {
            *filename++ = *url;
            count++;
        }
        if (count >= 95)
            break;
        url++;
    }
    filename = filename - count;
    strcat(filename, ".txt");
}
```

心得体会：

1. 本次对HTTP代理服务器的实现，理解了代理服务器的原理和执行过程，对Socket编程有了初步的了解，对HTTP请求和响应原理有了更深的认识；
2. 对TCP/IP协议有了更深的理解；
3. 对多线程优化下处理网络请求通过实际调试增加了理解；
4. 对网站钓鱼、网站屏蔽等有了深刻的理解。