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	袁野		院系	计算学部计算机科学与技术专业	
班级	1903102		学号	1190200122	
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	2021.10.31	
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：

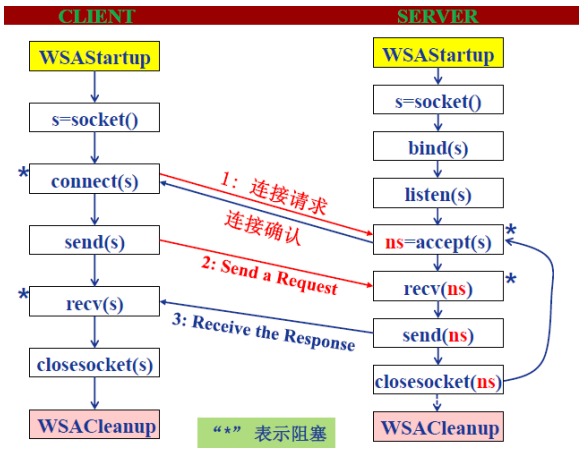
1. Socket 编程的客户端和服务器端主要步骤

1.1 客户端主要步骤

1. 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（三次握手）
2. 发送请求报文
3. 接收返回报文
4. 关闭连接

1.2 服务器端主要步骤

1. 创建套接字，绑定套接字的本地IP地址和端口号，对端口进行监听。
2. 从连接请求队列中取出一个连接请求，并同意连接。在TCP连接过程中进行了三次握手。
3. 接收客户端请求
4. 对请求进行响应，发送响应数据
5. 关闭连接



1.3 设置浏览器代理

以IE浏览器设置为例：打开浏览器工具浏览器选项——连接——局域网设置——代理服务器。设置地址为127.0.0.1，端口号为10240。

2. HTTP 代理服务器的基本原理

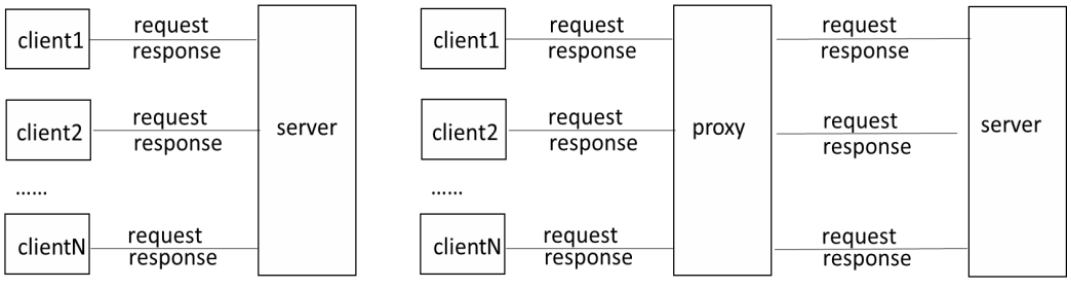
HTTP代理服务器用于一个网络终端（一般为客户端）通过代理服务与另一个网络终端（一般为服务器）进行非直接的连接。

2.1 主要功能

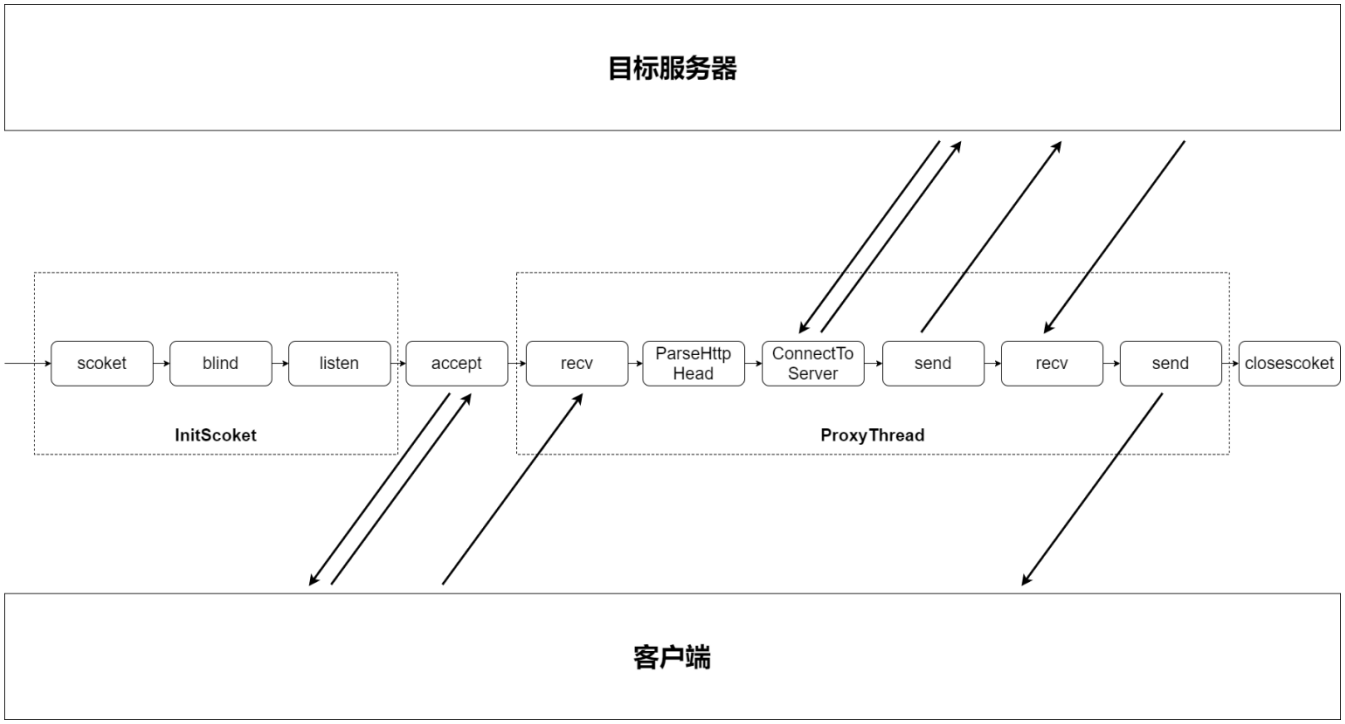
接收来自客户端的 HTTP 请求，并通过这个代理服务器将该请求转发给服务器；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端） 通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。

2.2 基本原理

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。



3. HTTP 代理服务器的程序流程图



4. 关键技术及解决方案

4.1. 关键技术：实现基本功能。

解决方案：参考代码中给出的以下函数

a) BOOL InitSocket()
首先加载套接字库，使用以下几个socket函数
socket(AF_INET, SOCK_STREAM, 0);
bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR));
和listen(ProxyServer, SOMAXCONN)
实现了服务器流程中的socket和bind和listen；

b) BOOL ParseHttpHead(char *buffer, HttpHeader * httpHeader)
对请求报文的头部文件进行解析，得到请求报文中的method, url, host和cookie等，用于ConnectToServer函数与目标服务器建立连接。

c) BOOL ConnectToServer(SOCKET *serverSocket, char *host)
使用socket创建套接字，connect连接至目标服务器

d) unsigned int __stdcall ProxyThread(LPVOID lpParameter)
实现了从客户端接收请求报文，向服务器发送请求报文，从服务器接收响应报文，向客户端送响应报文。

通过ParseHttpHead函数基对请求报文头部进行解析，然后将得到的头部文件作为ConnectToServer函数与目标服务器建立链接。连接成功后，便将请求报文发送过去，接收收到响应报文，然后发送响应报文给浏览器即可

4.2 关键技术：实现 cache 功能。

解决方案：在程序中创建一个类似httpHeader的结构体数组CACHE，用于储存http头部、报文内容、更

新时间等信息，每次访问时在CACHE中进行查找，如果找到与当前请求的http头部相等的报文请求记录，则对请求报文进行修改，添加is-modified标记用以查询该url内容最后一次修改的时间，若最后一次修改的时间与CACHE中记录的时间相同，则直接返回CACHE中的报文，否则重新请求报文并对CACHE中内容进行更新。具体函数如下所示。

```
struct cache_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
    cache_HttpHeader() {
        ZeroMemory(this, sizeof(cache_HttpHeader)); //内存置零
    }
};

struct __CACHE {
    cache_HttpHeader httphed;
    char buffer[MAXSIZE];
    char date[DATELENGTH];//存储的更新时间
    __CACHE() {
        ZeroMemory(this->buffer, MAXSIZE);
        ZeroMemory(this->date, DATELENGTH);
    }

};
```

```
//判断两个报文是否相同
BOOL Isequal(cache_HttpHeader htp1, HttpHeader htp2) {
    if (strcmp(htp1.method, htp2.method)) return false;
    if (strcmp(htp1.url, htp2.url)) return false;
    if (strcmp(htp1.host, htp2.host)) return false;
    return true;
}
```

```
//在缓存中找到对应的对象
int Cache_Search(__CACHE* cache, HttpHeader htp) {
    for (int i = 0; i < CACHE_MAXSIZE; ++i)
        if (Isequal(cache[i].httphed, htp)) return i;
    return -1;
}
```

a) BOOL Isequal(cache_HttpHeader htp1, HttpHeader htp2)
判断两个http头部是否相同

b) int Cache_Search(__CACHE* cache, HttpHeader htp)、
遍历CACHE中的所有已经缓存过的http头部并进行对比，如果相同则将其在CACHE中的序号返回，否则返回-1。

4.3 关键技术：实现用户过滤

解决方案：

用户过滤：记录所有的需要限制的用户IP，发送请求时检查发送请求的IP地址是否为需要限制的用户IP，如果不是则正常请求报文并返回，如果是则不进行请求；

```
if (RestricitiveTurnOn) {
    bool flag = 0;
    // printf("%s----------------------------------\n",inet_ntoa(verAddr.sin_addr));
    for (int i = 0; i < restrictiveClient_number; ++i) {
        if (!strcmp(restrictiveClient[i], inet_ntoa(verAddr.sin_addr))) {
            printf("用户访问受限\n");
            flag = 1;
            break;
        }
    }
    if (flag) continue;
}
```

4.4 关键技术：网站引导

解决方案：

判断发送的请求报文的url是否为引导页，如果是，则将报文内的url以及host均修改为被引导页，然后进行发送即可。

```
if (!strcmp(induceSite, httpHeader->host)) {
    printf("正在跳转网站……\n");
    Change(Buffer, httpHeader->host, targetSite);
    ZeroMemory(httpHeader->host, strlen(httpHeader->host));
    memcpy(httpHeader->host, targetSite, strlen(targetSite));
    sprintf_s(httpHeader->url, "http://%s/", targetSite);
}
```

4.5 关键技术：网站过滤

解决方案

在ParseHttpHead 解析 TCP 报文中的 HTTP 头部，将请求报文头部中的host与被禁止网站进行比较，如果出现相同的表示访问的网站被禁止访问。

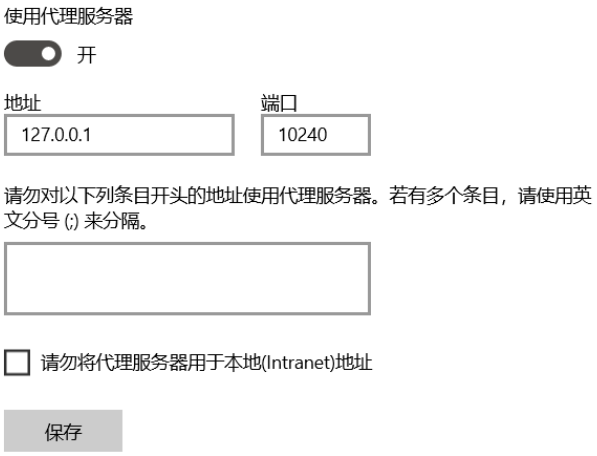
```
if (DisabledHostTurnOn) {
    for (int i = 0; i < disabledHost_number; ++i) {
        if (!strcmp(disabledHost[i], host)) {
            printf("该网站已被过滤\n");
            return false;
        }
    }
}
```

验证过程以及实验结果：

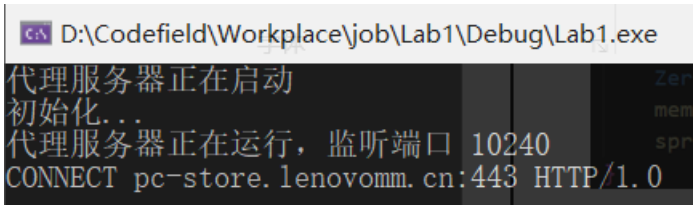
采用演示截图、文字说明等方式，给出本次实验的实验结果。

1. 实现了一个基本 HTTP 代理服务器

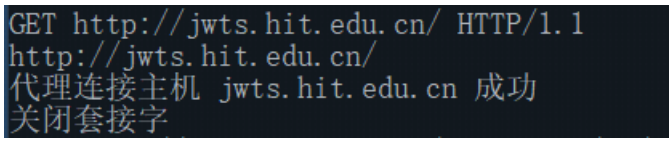
设置系统代理



运行代理服务器成功



以访问哈工大教务系统为例：

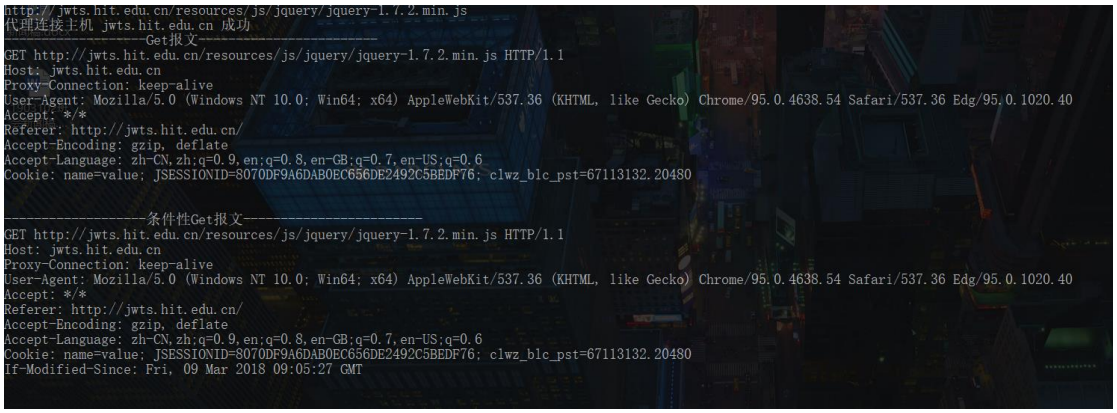


2. 实现一个支持 Cache 功能的 HTTP 代理服务器

再次访问jwts.hit.edu.cn

我们可以看到此时返回的报文是CACHE内储存的报文，内容如图所示。

如果我们将第二次访问的报文修改前与修改后输出，可以发现先区别在于最后多了If-Modified标记



3. 实现了以下扩展功能

a) 网站过滤：允许/不允许访问某些网站；

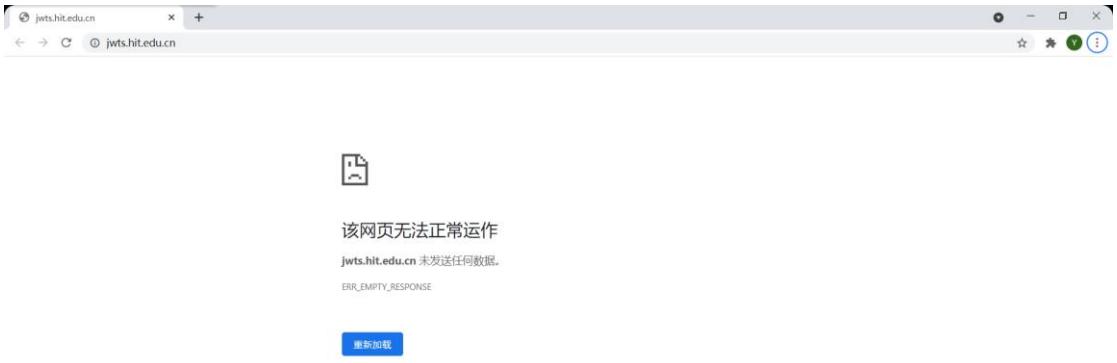
首先将过滤网站开关打开

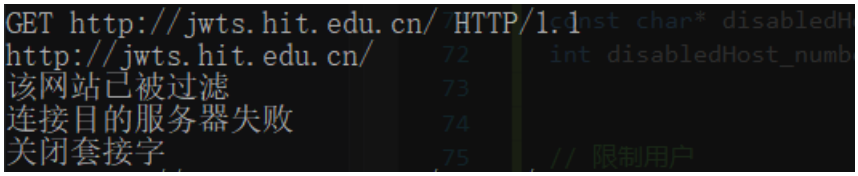
```
const bool DisabledHostTurnOn = true;
```

此时过滤列表有一个网站

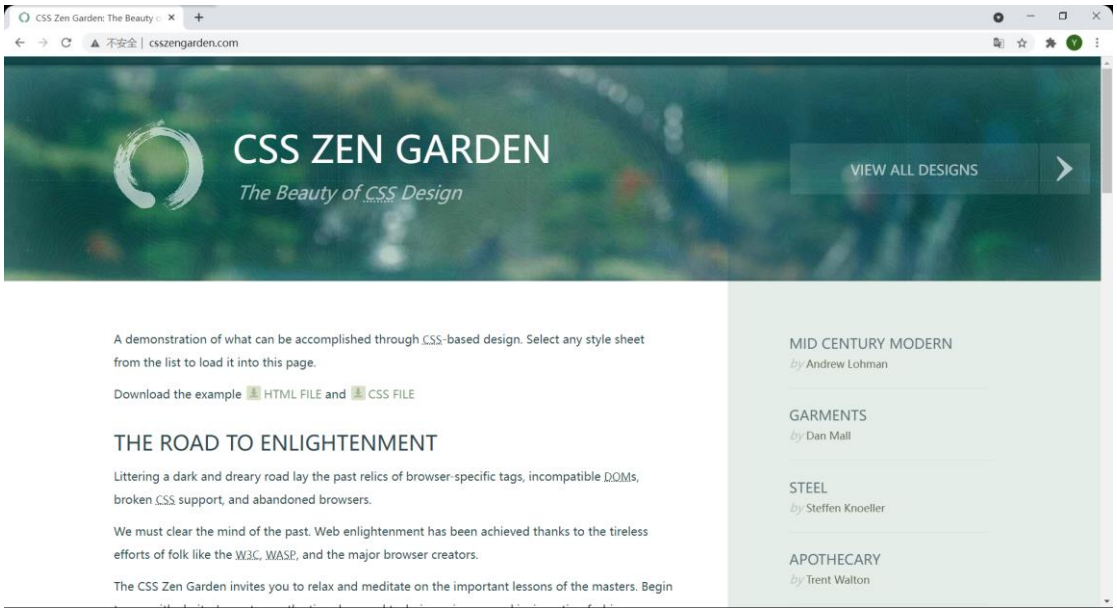
```
//禁用网站
const char* disabledHost[10] = { "jwts.hit.edu.cn" };
int disabledHost_number = 1;
```

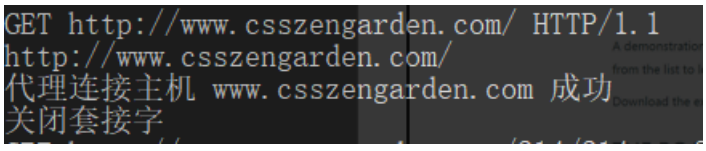
访问该网站





此时我们再访问其他网站



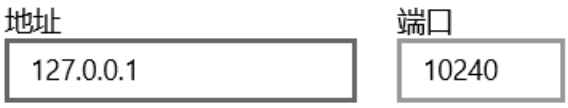


此时访问正常，说明其他网站并未受到影响。

b）用户过滤：支持/不支持某些用户访问外部网站；
首先将限制用户开关打开

```
const bool RestricitiveTurnOn = true;
```

此时IP地址为127.0.0.1

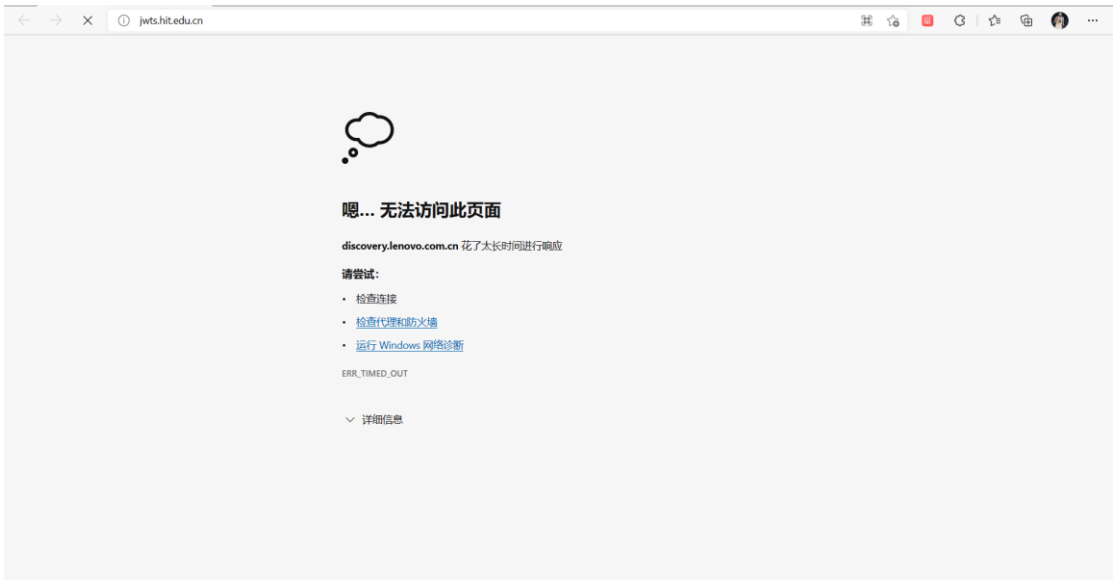


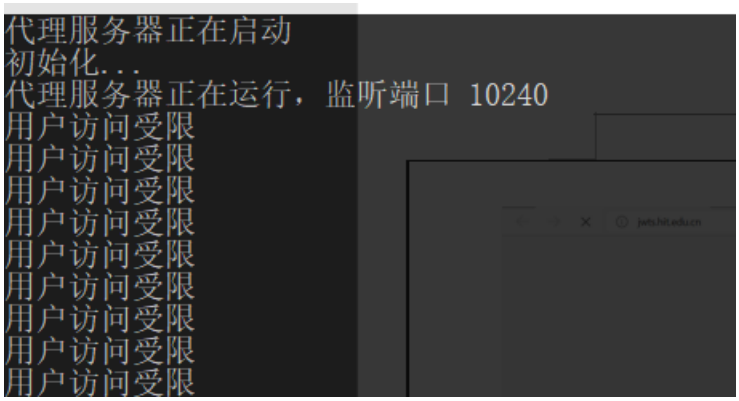
请勿对以下列各目开头的地址使用代理服务器，若有多

限制用户列表为

```
// 限制用户
const char* restrictiveClient[10] = { "127.0.0.1" };
int restrictiveClient_number = 1;
```

此时我们再试图访问一些网站

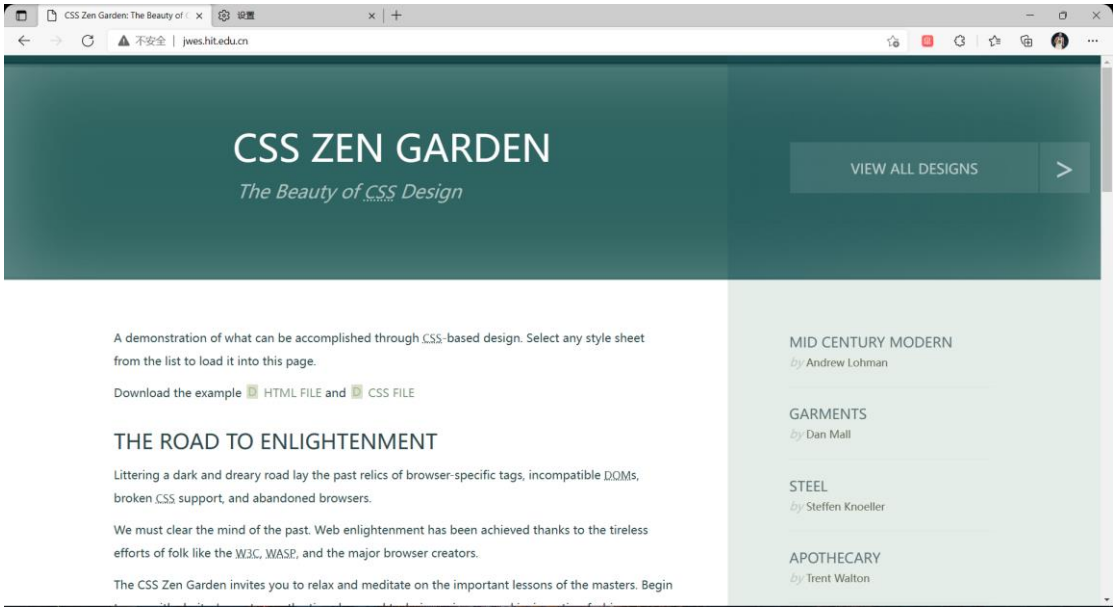




c）网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

我们试图将对于jwes.hit.edu.cn的访问引导至www.csszengarden.com

我们发现虽然访问的网址为jwes.hit.edu.cn，但是展示出来的界面却是www.csszengarden.com的界面



问题讨论：

1. 问题：添加If-Modified并发送之后总是反馈报文格式不对。
   原因：报文最后有两个空行，即两个“\r\n”。修改后发送报文的反馈正常。
2. 有时添加If-Modified之后报文中会出现两个If-Modified
   原因：浏览器本身的cache储存报文之后会在最后添加一个If-Modified请求，我们的代理服务器再添加一个If-Modified请求，就会有两个相同请求，因此我们需要先判断是否已经包含该请求，同时浏览器缓存会对我们的代理产生许多其他的影响，因此我在每次访问之后都会清理一次浏览器缓存。
3. 再试图钓鱼时，修改Host之后钓鱼失败
   原因：观察报文之后发现，除了修改Host，还需要修改url为对应的网址，然后钓鱼成功。

心得体会：

经过此次实验，熟悉了Socket 网络编程，清楚客户端和服务器之间Socket通信过程；掌握了HTTP 代理服务器的基本工作原理；同时了解了钓鱼网站，禁止用户，禁止网站以及 Cache等的原理。让我对网络编程更感兴趣。

HTTP 代理服务器源代码（带有详细注释）

```
#include <stdio.h>
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>
#include <tchar.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 CACHE_MAXSIZE 100   //最大缓存数
#define DATELENGTH 50          //时间字节数

const bool RestricitiveTurnOn = false;   // 限制用户访问开关
const bool DisabledHostTurnOn = false;   // 网址过滤开关
// 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));
	}
};

struct cache_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
	cache_HttpHeader() {
		ZeroMemory(this, sizeof(cache_HttpHeader)); //内存置零
	}
};

struct __CACHE {
	cache_HttpHeader httphed;
	char buffer[MAXSIZE];
	char date[DATELENGTH];//存储的更新时间
```

```
    __CACHE() {
        ZeroMemory(this->buffer, MAXSIZE);
        ZeroMemory(this->date, DATELENGTH);
    }

};
int Cache_index = 0;//标记下一个应该放缓存的位置
__CACHE cache[CACHE_MAXSIZE];

const char* ife = "If-Modified-Since: ";
const char* blank = " ";
const char* Modd = "304";


BOOL InitSocket();
void ParseHttpHead(char* buffer, HttpHeader* httpHeader);
BOOL ConnectToServer(SOCKET* serverSocket, char* host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
int Cache_Search(__CACHE* cache, HttpHeader htp);
void Change(char* res, char* a, const char* b);

//代理相关参数
SOCKET ProxyServer;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 10240;
//禁用网站
const char* disabledHost[10] = { "jwts.hit.edu.cn" };
int disabledHost_number = 1;


//  限制用户
const char* restrictiveClient[10] = { "127.0.0.1" };
int restrictiveClient_number = 1;

//网站诱导
const char* induceSite = "jwes.hit.edu.cn";
const char* targetSite = "www.csszengarden.com";
//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};
struct ProxyParam {
    SOCKET clientSocket;
```

```
    SOCKET serverSocket;
};
int _tmain(int argc, _TCHAR* 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;
    ProxyParam* lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;
    //代理服务器不断监听
    sockaddr_in verAddr;
    int dds = sizeof(SOCKADDR);
    while (true) {
        acceptSocket = accept(ProxyServer, (SOCKADDR*)&verAddr, &(dds));
        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL) {
            continue;
        }
        if (RestricitiveTurnOn) {
            bool flag = 0;
            // printf("%s----------------------------------\n",inet_ntoa(verAddr.sin_addr));
            for (int i = 0; i < restrictiveClient_number; ++i) {
                if (!strcmp(restrictiveClient[i], inet_ntoa(verAddr.sin_addr))) {
                    printf("用户访问受限\n");
                    flag = 1;
                    break;
                }
            }
            if (flag) continue;
        }

        lpProxyParam->clientSocket = acceptSocket;
        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;
    //版本 2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载 dll 文件 Scoket 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;
    }
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
        printf("不能找到正确的 winsock 版本\n");
        WSACleanup();
        return FALSE;
    }
    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;
    }
    ProxyServerAddr.sin_family = AF_INET;
    ProxyServerAddr.sin_port = htons(ProxyPort);
    ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    // ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//设置 IP 地址
    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;
}
//**********************************
// Method: ProxyThread
// FullName: ProxyThread
// Access: public
// Returns: unsigned int __stdcall
// Qualifier:  线程执行函数
// Parameter: LPVOID lpParameter
//**********************************
unsigned int __stdcall ProxyThread(LPVOID lpParameter) {
        const char* delim = "\r\n";
        char Buffer[MAXSIZE];
        char* CacheBuffer;
        char* cacheBuff;
        char* ptr;
        char ThisDate[DATELENGTH];
        char* p;
        ZeroMemory(Buffer, MAXSIZE);
        SOCKADDR_IN clientAddr;
        int recvSize;
        int ret;
        int index;
        bool isUpdate;
        recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
        HttpHeader* httpHeader = new HttpHeader();
        if (recvSize <= 0) {
            printf("本地客户端等待中\n");
            goto error;
        }
        CacheBuffer = new char[recvSize + 1];
        ZeroMemory(CacheBuffer, recvSize + 1);
        memcpy(CacheBuffer, Buffer, recvSize);
        ParseHttpHead(CacheBuffer, httpHeader);
        delete CacheBuffer;

        if (!strcmp(induceSite, httpHeader->host)) {
            printf("正在跳转网站……\n");
            Change(Buffer, httpHeader->host, targetSite); //  将报文中所有的原网址替换为新网
址
            ZeroMemory(httpHeader->host, strlen(httpHeader->host));
```

```
            memcpy(httpHeader->host, targetSite, strlen(targetSite));
            sprintf_s(httpHeader->url, "http://%s/", targetSite);// 将新报文的 url 修改
        }

        // printf("%s--------\n%s------------------\n", httpHeader->host, httpHeader->url);

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

        index = Cache_Search(cache, *httpHeader); // 在 CACHE 中寻找是否缓存过该报文
        //                      printf("-------------------------\n%s\n%s\n%s\n--------------------------------
\n",httpHeader->host,httpHeader->url,httpHeader->method);
        // printf("%d ----- \n",index);
        ZeroMemory(ThisDate, DATELENGTH);

        if (index >= 0) {
            memcpy(ThisDate, cache[index].date, strlen(cache[index].date));
            char ThisBuffer[MAXSIZE];
            ZeroMemory(ThisBuffer, MAXSIZE);
            // printf("-------------------Get 报文-----------------------\n%s\n", Buffer);
            p = strtok_s(Buffer, delim, &ptr);
            bool flag = false; // 判断是否已经有 If-Modified 请求
            while (p) {
                // printf("------------%s\n",p);
                int length = strlen(ThisBuffer);
                memcpy(ThisBuffer + length, p, strlen(p));
                memcpy(ThisBuffer + length + strlen(p), delim, strlen(delim));
                if (p[0] == 'I') {
                    char now[100];
                    ZeroMemory(now, 100);
                    memcpy(now, p, 11);
                    if (!memcmp(now, "If-Modified", 11)) {
                        flag = true;
                    }
                }
                p = strtok_s(NULL, delim, &ptr);
            }
            if (!flag) { // 添加 If-Modified 请求
                int length = strlen(ThisBuffer);
                memcpy(ThisBuffer + length, ife, strlen(ife));
```

```
                length += strlen(ife);
                memcpy(ThisBuffer + length, cache[index].date, strlen(cache[index].date));
                length = strlen(ThisBuffer);
                memcpy(ThisBuffer + length, delim, strlen(delim));
            }
            int length = strlen(ThisBuffer);
            memcpy(ThisBuffer + length, delim, strlen(delim));
            //将客户端发送的 HTTP 数据报文处理后转发给目标服务器
            // printf("--------------------条件性 Get 报文------------------------\n%s\n", ThisBuffer);
            ret = send(((ProxyParam*)lpParameter)->serverSocket, ThisBuffer, strlen(ThisBuffer)
+ 1, 0);
            // printf("--------------------------------------------\n%s\n", ThisBuffer);
            // printf("------------------------------------------------\n");
            //等待目标服务器返回数据
            recvSize = recv(((ProxyParam*)lpParameter)->serverSocket, ThisBuffer, MAXSIZE,
0);
            // printf("------------------Server 返回报文-------------------\n%s\n", ThisBuffer);
            // printf("--------------------------------------------\n");

            if (recvSize <= 0) {
                goto error;
            }

            if (!memcmp(&ThisBuffer[9], Modd, strlen(Modd))) {
                ret = send(((ProxyParam*)lpParameter)->clientSocket, cache[index].buffer,
strlen(cache[index].buffer) + 1, 0);
                printf("-----------------------------------\n");
                printf("\n 将 cache 中的数据返回给客户端\n\n");
                printf("-----------------------------------\n");
                goto error;
            }
            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) {
            printf("目标服务器未响应\n");
            goto error;
        }
```

```
    cacheBuff = new char[MAXSIZE];
    ZeroMemory(cacheBuff, MAXSIZE);
    memcpy(cacheBuff, Buffer, MAXSIZE);

    ZeroMemory(ThisDate, sizeof(ThisDate));
    p = strtok_s(cacheBuff, delim, &ptr);
    isUpdate = false;
    while (p) { // 提取该报文的 Last-Modified 的时间
        if (strlen(p) > 15) {
            char header[15];
            ZeroMemory(header, sizeof(header));
            memcpy(header, p, 14);
            if (!(strcmp(header, "Last-Modified:"))) {
                memcpy(ThisDate, &p[15], strlen(p) - 15);
                isUpdate = true;
                break;
            }
        }
        p = strtok_s(NULL, delim, &ptr);
    }
    delete(cacheBuff);

    //如果有更新，将新的报文放到缓存里
    if (isUpdate) {
        if (index >= 0) {
            memcpy(&(cache[index].buffer), Buffer, strlen(Buffer));
            memcpy(&(cache[index].date), ThisDate, strlen(ThisDate));
        }
        else {
            //循环使用
            memcpy(&(cache[Cache_index % CACHE_MAXSIZE].httphed.host),
httpHeader->host, strlen(httpHeader->host));
            memcpy(&(cache[Cache_index % CACHE_MAXSIZE].httphed.method),
httpHeader->method, strlen(httpHeader->method));
            memcpy(&(cache[Cache_index % CACHE_MAXSIZE].httphed.url),
httpHeader->url, strlen(httpHeader->url));
            memcpy(&(cache[Cache_index % CACHE_MAXSIZE].buffer), Buffer,
strlen(Buffer));
            memcpy(&(cache[Cache_index % CACHE_MAXSIZE].date), ThisDate,
strlen(ThisDate));
            Cache_index++;
        }
    }
```

```
        // printf("-----------------------------%d\n",Cache_index++);

        //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
//************************************
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);
        }
        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);
        }
}
//************************************
// 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);

    if (DisabledHostTurnOn) {
        for (int i = 0; i < disabledHost_number; ++i) {
            if (!strcmp(disabledHost[i], host)) { // 与过滤列表中的网址进行一一比对
                printf("该网站已被过滤\n");
                return false;
            }
        }
    }

    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));
    *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;
}

//判断两个报文是否相同
BOOL Isequal(cache_HttpHeader htp1, HttpHeader htp2) {
    if (strcmp(htp1.method, htp2.method)) return false;
    if (strcmp(htp1.url, htp2.url)) return false;
    if (strcmp(htp1.host, htp2.host)) return false;
    return true;
}

//在缓存中找到对应的对象
int Cache_Search(__CACHE* cache, HttpHeader htp) {
    for (int i = 0; i < CACHE_MAXSIZE; ++i)
        if (Isequal(cache[i].httphed, htp)) return i;
    return -1;
}

// 将 res 中的所有 a 字符串修改为 b 字符串

void Change(char* res, char* a, const char* b) {
    char ret[MAXSIZE];
    ZeroMemory(ret, sizeof(ret));
    int p = 0;
    int length = 0;
    while (p < (int)strlen(res)) {
        if (!memcmp(res + p, a, strlen(a))) {
            memcpy(ret + length, b, strlen(b));
            ret[length += strlen(b)] = '\0';
            p += strlen(a);
            continue;
        }
```

```
            ret[length++] = res[p++];
            ret[length] = '\0';
        }
        ret[strlen(ret)] = '\0';
        memcpy(res, ret, strlen(ret));
}
```