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	冯开来		院系	计算学部	
班级	1903602		学号	1190201215	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物 207		实验时间	2021.10.30	
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. 主函数

整个过程可以理解为，我们自己写的代理服务器，收到一个客户端（其实还是我们自己）的http请求，我们这个服务器对这个请求分析后再发送到目的网址。其中收到和发送这个请求会用到Socket的相关知识。所以大致流程是先初始化代理服务器的接收端Socket和发送端Socket（其中接收端Socket实质上就是表示客户主机的Socket），然后使代理服务器不断监听。

```
int main(int argc, char* argv[])
{
    printf("代理服务器正在启动\n");
    printf("初始化...\n");

    // 将本地地址和套接字绑定，并监听套接字的链接请求
    // 套接字设置为监听模式
    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;
        }
```

2. 初始化Socket

a) 初始化套接字库

这一部分的主要实现的是初始化代理服务器的Socket。如果初始化成功返回TRUE。将

本地的端点地址绑定到套接字上。这一段代码有参考代码，所以不过多说明。

```
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); // 将两个byte型合成一个word型
    //加载dll 文件Scoket 库  同操作系统说明，我们需要哪个库文件，让该库文件与当前应用程序绑定
    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;
    }
```

b)　创建Socket

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Internet通信;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；

```
    //AF_INET,PF_INET IPV4 Internet协议
    //SOCK_STREAM Tcp连接，提供序列化可靠双向连接
    ProxyServer = socket(AF_INET, SOCK_STREAM, 0); // IPV4地址族，流套接字，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

    //将整型变量从主机字节顺序转变成网络字节顺序
    ProxyServerAddr.sin_port = htons(ProxyPort);



    //ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    // 将一个点分十进制的IP转换成一个长整数型数（u_long类型）
    ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");

    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;
}
```

3. 代理服务器不断监听

这一过程就是一直有一个欢迎套接字acceptSocket来接收来自客户端的请求，然后为每一个请求创建一个线程，执行代理过程。最后关闭代理服务器，关闭线程。

```
    //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，执行一对一的代理过程
        hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0);

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

4. 创建线程函数

调用的线程函数是这样的：

```
// 创建子线程，执行一对一的代理过程
hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0);
```

在线程函数中，首先需要初始化缓存，请求报文中url名。

```
char Buffer[MAXSIZE];
char* CacheBuffer;
ZeroMemory(Buffer, MAXSIZE); // 用0来填充一块区域

char fileBuffer[MAXSIZE];

char filename[100];

HttpHeader* httpHeader = new HttpHeader();


// sockaddr_in ?
SOCKADDR_IN clientAddr;
int length = sizeof(SOCKADDR_IN);
int recvSize;
int ret;

recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);

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

接收客户端请求消息，与其通信，调用recv()函数接受客户端请求的消息。函数返回值为实际收到的消息字节数，消息内容缓存在Buffer中。

```
recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
```

然后将Buffer拷贝一份到CacheBuffer中，开始解析http头部。

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

得到的信息就是http请求报文的头部，如下图：

然后我们可以通过这些信息完成下面的附加功能（下一节内容会详细说明）。之后的工作就是是否能连接到我们需要访问的网址，调用ConnectToServer函数。这个函数会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连接，如果连接成功，放回TRUE。

```
309 BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
310     sockaddr_in serverAddr;
311     serverAddr.sin_family = AF_INET;
312     serverAddr.sin_port = htons(HTTP_PORT);
313     HOSTENT* hostent = gethostbyname(host);
314     if (!hostent) {
315         return FALSE;
316     }
317     in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
318     serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
319     *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
320     if (*serverSocket == INVALID_SOCKET) {
321         return FALSE;
322     }
323     if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr))
324         == SOCKET_ERROR) {
325         closesocket(*serverSocket);
326         return FALSE;
327     }
328     return TRUE;
329 }
```

下一步就是将客户端发送的HTTP请求报文转发给目标服务器，这一步也是调用函数send实现。send()函数是发送一次数据，返回值为成功发送的字节数，该值可能会小于需要发送的字节数；sendall()函数则会在一次未全部发送完之后尝试发送剩下的所有字节，成功返回None，失败则抛出异常；

```
//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);
```

接下来等待目标服务器返回数据，可以理解为网页内容，调用recv()函数。

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

代理服务器将数据直接转发给客户机。调用send()函数。

```
//将目标服务器返回的数据直接转发给客户端
// 第一个发送到socket， 第二个要发的东西，第三个实际发送的东西，返回实际发送的字节
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;
}
```

5. 附加功能实现

a) 实现缓存及缓存更新

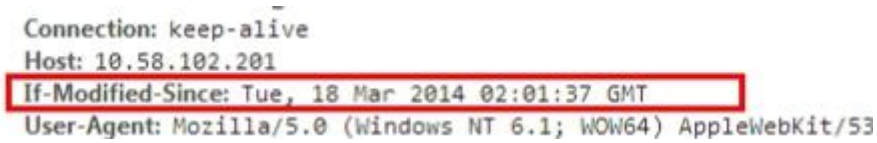
我们首先要实现缓存，在目标服务机将数据发送给我们代理服务器之后，我们检查我们是否已有缓存，如果有，则打开缓存文件，更新缓存。如果没有就新建一个缓存文件。

```
// 是否有缓存，一般来说false
if (haveCache)
{
    getCache(Buffer, filename);
}
else {
    makeCache(Buffer, httpHeader->url);  //缓存报文
}
```

新建缓存的文件过程首先拷贝头部信息，检查头部状态码是否是200，只有200说明是成功第一次访问的。然后将整个http数据报写入文件，文件名为目的服务器的url。

```
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(tempBuffer, delim);//提取第一行
    memcpy(num, &p[9], 3);

    if (strcmp(num, "200") == 0) {  //状态码是200时缓存
        // 200指成功访问，404就是没成功

        // 构建文件
        char filename[100];
        ZeroMemory(filename, 100);
        makeFilename(url, filename);
        printf("filename : %s\n", filename);

        FILE* out;
        out = fopen(filename, "w");
        fwrite(buffer, sizeof(char), strlen(buffer), out);
        fclose(out);
        printf("\n**********************************\n\n");
        printf("\n网页已经被缓存\n");
    }
}
```

接下来考虑缓存更新问题，因为如果不是第一次访问目的网页的话，那么http请求报文就需要进行修改，在头部行加入if-modified-since。加入的具体位置如下图：



所以我们就要判断哪个请求报文是需要加入if-modified-since的，这里的方法是根据请求报文中的目的主机名，通过gethostbyname()函数得到目的主机的url，然后再文件中寻找是否有相同的文件，如果有说明之前已经有过缓存那么就是说明我们不是第一次访问该目的网址。

```
// 是否已经有缓存
if ((in = fopen(filename, "rb")) != NULL)
{
    fread(fileBuffer, sizeof(char), MAXSIZE, in);
    fclose(in);

    getDate(fileBuffer, field, date_str);

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

然后我们使用makeNewHTTP()函数，加入if-modified-since函数，得到新的http请求报文，再发送给目的网址。

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

之后，再目标服务器返回http数据之后，我们需要对新的数据进行缓存，就是要修改之前的缓存报文。这里再makeNewHttp函数结束后，将标志位haveCache置为TRUE，这样我们在后续中不会调用makeCache()函数，而是调用getCache()函数对已经缓存的文件进行修改。

```
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(tempBuffer, delim);//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("\n************************************\n\n");
        printf("从本机获得缓存\n");
        ZeroMemory(buffer, strlen(buffer));
        FILE* in = NULL;
        if ((in = fopen(filename, "r")) != NULL) {
            fread(buffer, sizeof(char), MAXSIZE, in);
            fclose(in);
        }
    }
}
```

b) 屏蔽特定网站

这个功能很简单，就是在一开始得到目的网址的url后，检查是否和我们想要屏蔽的网址是否相同，如果相同，直接跳转到error。

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

c) 允许特定用户访问

这一部分也是通过http请求报文的头部信息中的host名称，然后通过函数gethostbyname()得到我们客户机的IP地址，进行匹配，如果和我们需要屏蔽的IP地址，就直接用户过滤。

```
//过滤功能
if (!strcmp(forbiduser, inet_ntoa(caddr.sin_addr)))
{
    printf("用户过滤\n");
    continue;
}
```

d) 实现钓鱼

实现钓鱼也非常简单，匹配到和钓鱼原网址相同的网址，我们修改http头部的目的网址的ip地址和端口号，修改目的主机名，然后由代理服务器发送修改后的请求报文。

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

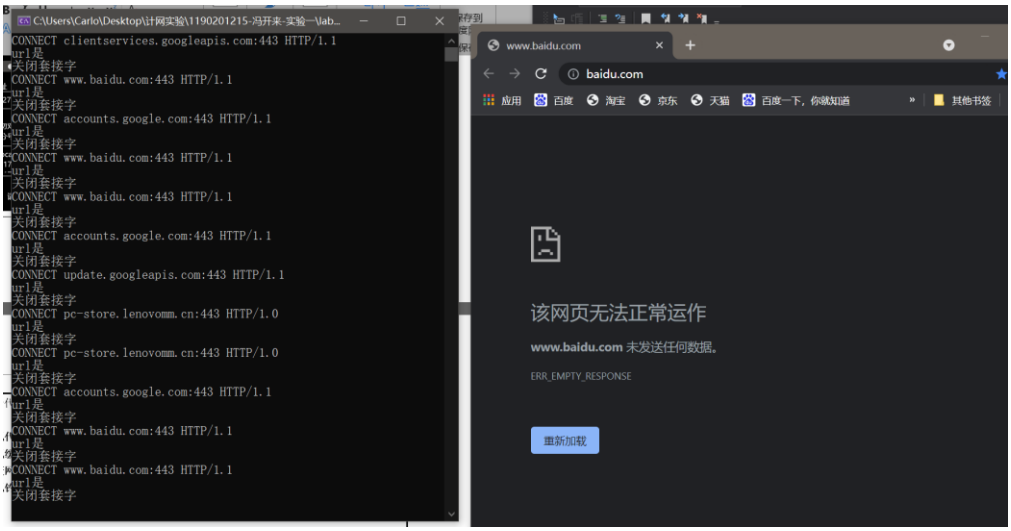
实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

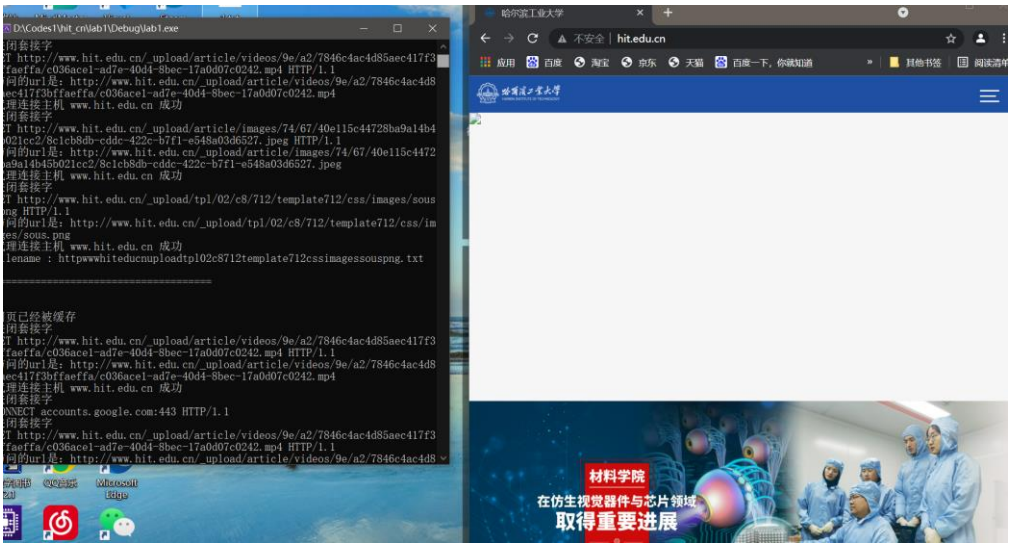
1. **启动程序**

打开代理，设置好ip地址和端口号

运行代码，这时候正常https协议的网站是无法访问的：

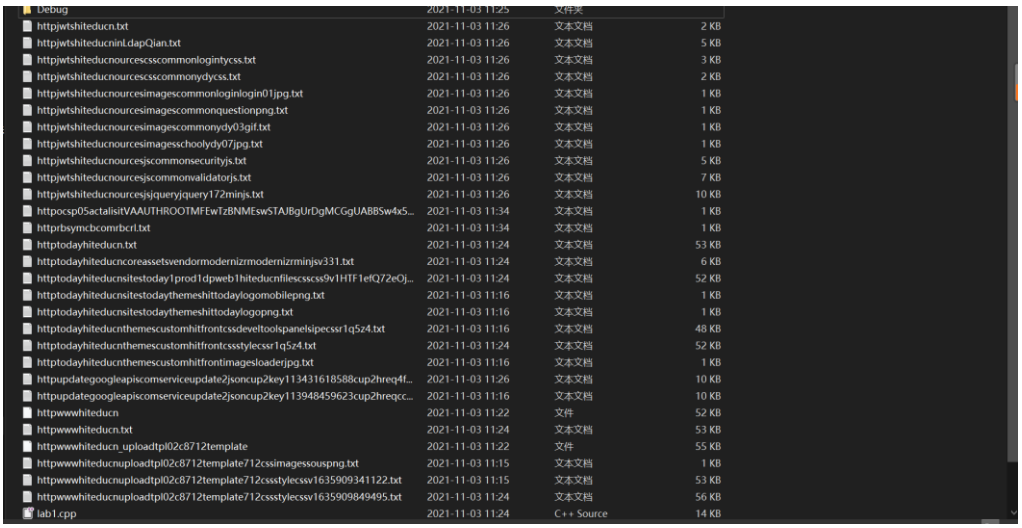


2. **实现代理服务器功能，访问哈工大校园网**



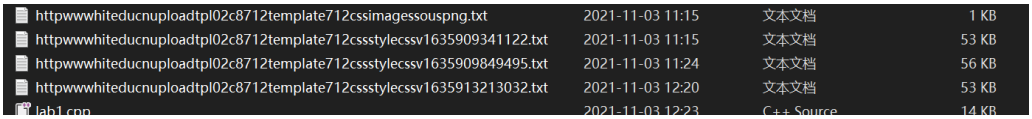
3. **实现缓存，并更新缓存文件**

缓存文件



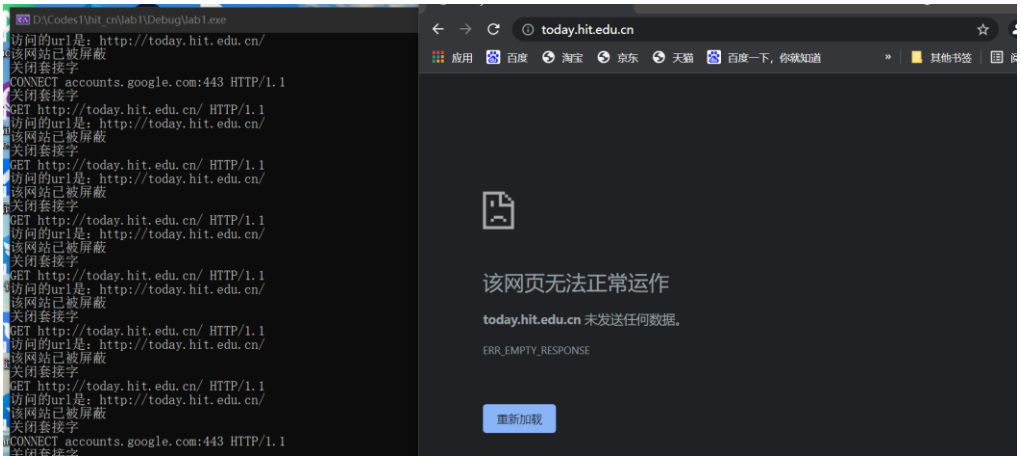
更新缓存，再次登录官网，可以看到缓存已经更新成功

更新后的缓存文件：



4. **屏蔽网站今日哈工大**

可以看到提示，该网站已被屏蔽



5. **屏蔽特定用户**

本次屏蔽用户的主机是127.0.0.1。可以看到提示该用户已被过滤：

6. **实现钓鱼，钓鱼源网址是4399小游戏，钓鱼后是哈工大教务处网站**

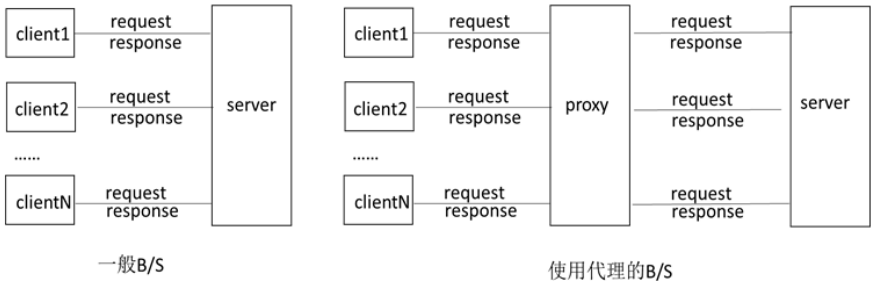
但是为什么没有加载出图片，应该是没有拦截后续请求图片的报文

问题讨论：

因为以下问题在上文以及有所讨论，所以接下来内容为精简概述：

1. Socket编程的客户端和服务端主要步骤
   a) 客户端
      - 初始化套接字库
      - 创建Socket
      - 向服务器发出连接请求
      - 连接建立后，向服务器请求数据，并置于等待状态，等待服务器返回数据
      - 关闭连接
      - 关闭套接字库
   b) 服务端
      - 初始化套接字库
      - 创建套接字
      - 绑定套接字
      - 监听端口
      - 接受连接请求，返回新的套接字
      - 接受客户端请求消息，返回请求数据，与其通信
      - 关闭套接字
      - 关闭套接字库
2. HTTP代理服务器原理

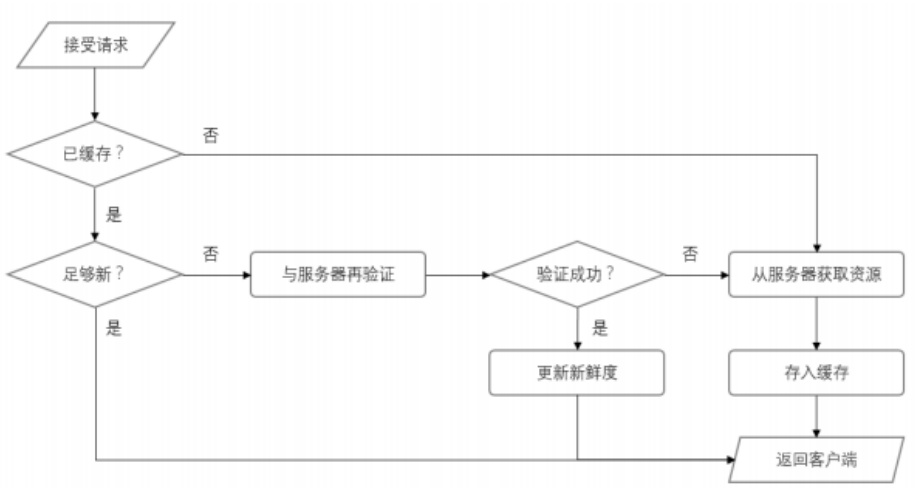
代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。如下图所示，为普通Web应用通信方式与采用代理服务器的通信方式的对比。



Web应用通信方式对比

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。

6. HTTP代理服务器源代码

```
//#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 FISH_WEB_SRC "http://www.4399.com/"     //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;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 8080; // 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 makeCache(char* buffer, char* url);
void getCache(char* buffer, char* filename);
void makeNewHTTP(char* buffer, char* value);
void getDate(char* buffer, char* field, char* tempDate);
void makeFilename(char* url, char* filename);
```

```
//***********************************
// 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); // 将两个byte型合成一个word型
    //加载dll 文件Scoket 库  向操作系统说明，我们需要哪个库文件，让该库文件与当前应用程序绑定
    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,PF_INET IPV4 Internet协议
    //SOCK_STREAM Tcp连接，提供序列化可靠双向连接
    ProxyServer = socket(AF_INET, SOCK_STREAM, 0); // IPV4地址族，流套接字，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
```

```
        //将整型变量从主机字节顺序转变成网络字节顺序
        ProxyServerAddr.sin_port = htons(ProxyPort);



        //ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
        // 将一个点分十进制的IP转换成一个长整数型数（u_long类型）
        ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");

        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
    // //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) {

        BOOL haveCache = false;
        BOOL needCache = true;

        char Buffer[MAXSIZE];
        char* CacheBuffer;
        ZeroMemory(Buffer, MAXSIZE); // 用0来填充一块区域

        char fileBuffer[MAXSIZE];

        char filename[100];
```

```
        HttpHeader* httpHeader = new HttpHeader();


        // sockaddr_in ?
        SOCKADDR_IN clientAddr;
        int length = sizeof(SOCKADDR_IN);
        int recvSize;
        int ret;

        recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE,
0);

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

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


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

        FILE* in;

        makeFilename(httpHeader->url, filename);

        char* field;
        field = (char*)"Date";

        char date_str[30];
        ZeroMemory(date_str, 30);

        // 是否已经有缓存
        if ((in = fopen(filename, "rb")) != NULL)
        {
            fread(fileBuffer, sizeof(char), MAXSIZE, in);
            fclose(in);
```

```
                getDate(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("\n**********该网站已被屏蔽**********\n");
                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("\n*******目标网址已被引导*******\n");
                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);



            //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;
            }
            // 是否有缓存，一般来说false
```

```
        if (haveCache)
        {
            getCache(Buffer, filename);
        }
        else {
            makeCache(Buffer, httpHeader->url);  //缓存报文
        }

        //将目标服务器返回的数据直接转发给客户端
        // 第一个发送到socket， 第二个要发的东西，第三个实际发送的东西，返回实际发送的字节
        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";

        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);
	}
	//else if (p[0] == 'C') {
	//	// connect
	//	return false;
	//}

	printf("url是%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);
	}
	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));
	*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 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(tempBuffer, delim);//提取第一行
	memcpy(num, &p[9], 3);

	if (strcmp(num, "200") == 0) {	//状态码是200时缓存
		// 200指成功访问，404就是没成功

		// 构建文件
		char filename[100];
```

```
                ZeroMemory(filename, 100);
                makeFilename(url, filename);
                printf("filename : %s\n", filename);

                FILE* out;
                out = fopen(filename, "w");
                fwrite(buffer, sizeof(char), strlen(buffer), out);
                fclose(out);
                printf("\n************************************\n\n");
                printf("\n网页已经被缓存\n");
            }
        }

        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(tempBuffer, delim);//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("\n************************************\n\n");
                printf("从本机获得缓存\n");
                ZeroMemory(buffer, strlen(buffer));
                FILE* in = NULL;
                if ((in = fopen(filename, "r")) != NULL) {
                    fread(buffer, sizeof(char), MAXSIZE, in);
                    fclose(in);
                }
            }
        }

        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);
```

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

    void getDate(char* buffer, char* field, char* tempDate) {
        char* p, * ptr, temp[5];
        ZeroMemory(temp, 5);
        //*field = "If-Modified-Since";

        const char* delim = "\r\n";
        p = strtok(buffer, delim); //  按行读取
        //printf("%s\n", p);
        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);
                // printf("tempDate: %s\n", tempDate);
            }
            p = strtok(NULL, delim);
        }
    }

    void makeFilename(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++;
            }
            strcat(filename, ".txt");
        }


        //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");

            // 将本地地址和套接字绑定，并监听套接字的链接请求
            // 套接字设置为监听模式
            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，执行一对一的代理过程
            hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread,
(LPVOID)lpProxyParam, 0, 0);

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

心得体会：

1. 本次对 HTTP 代理服务器的实现，更加理解了代理服务器的原理和执行过程；
2. 同时感受到了多线程执行的优化；
3. 同时了解到了 hashlib 的摘要算法，更加清晰了 socket 库中实现 HTTP 连接的各类函数及其作用；