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	郑旭然		院系	软件工程	
班级	2037102		学号	120L020719	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物楼 207		实验时间	2022.9.28	
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

客户端软件流程

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）
5. 关闭连接（closesocket）

服务器端软件流程

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）
5. 关闭连接（closesocket）

对于UDP协议上的通信，无需提前建立连接，只需在开始时建立相应的socket，进入无限循环，接收消息后直接与源地址进行通信即可。对于TCP协议上的通信，服务器需要有一个socket负责控制，在进入无限循环前建立绑定指定的端口号，并在无限循环内，对于每一个连接新建TCP连接与源主机进行通信即可。

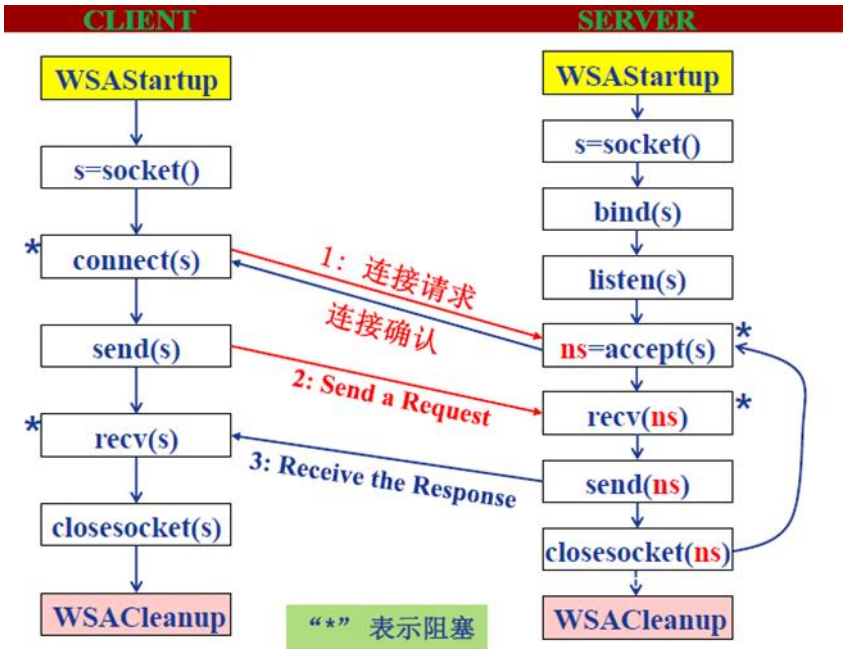


图 1 网络应用的 Socket API(TCP)调用基本流程

2. HTTP 代理服务器的基本原理

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。如图 1-2 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。

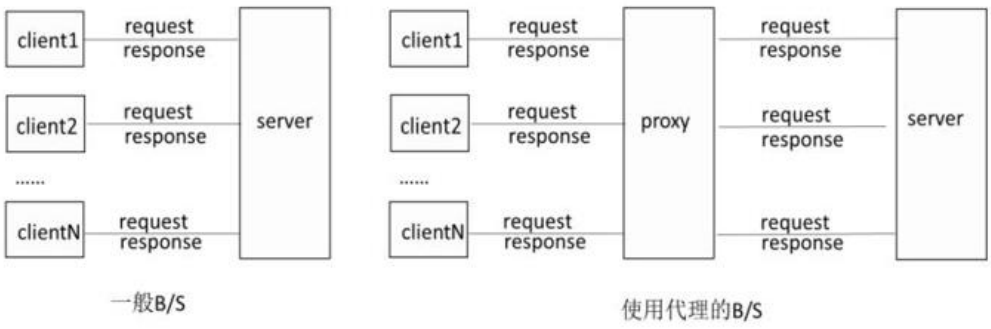


图 2 Web 应用通信方式对比

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。

我们实验中设计的是多用户代理服务器。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。

3. HTTP 代理服务器的程序流程图

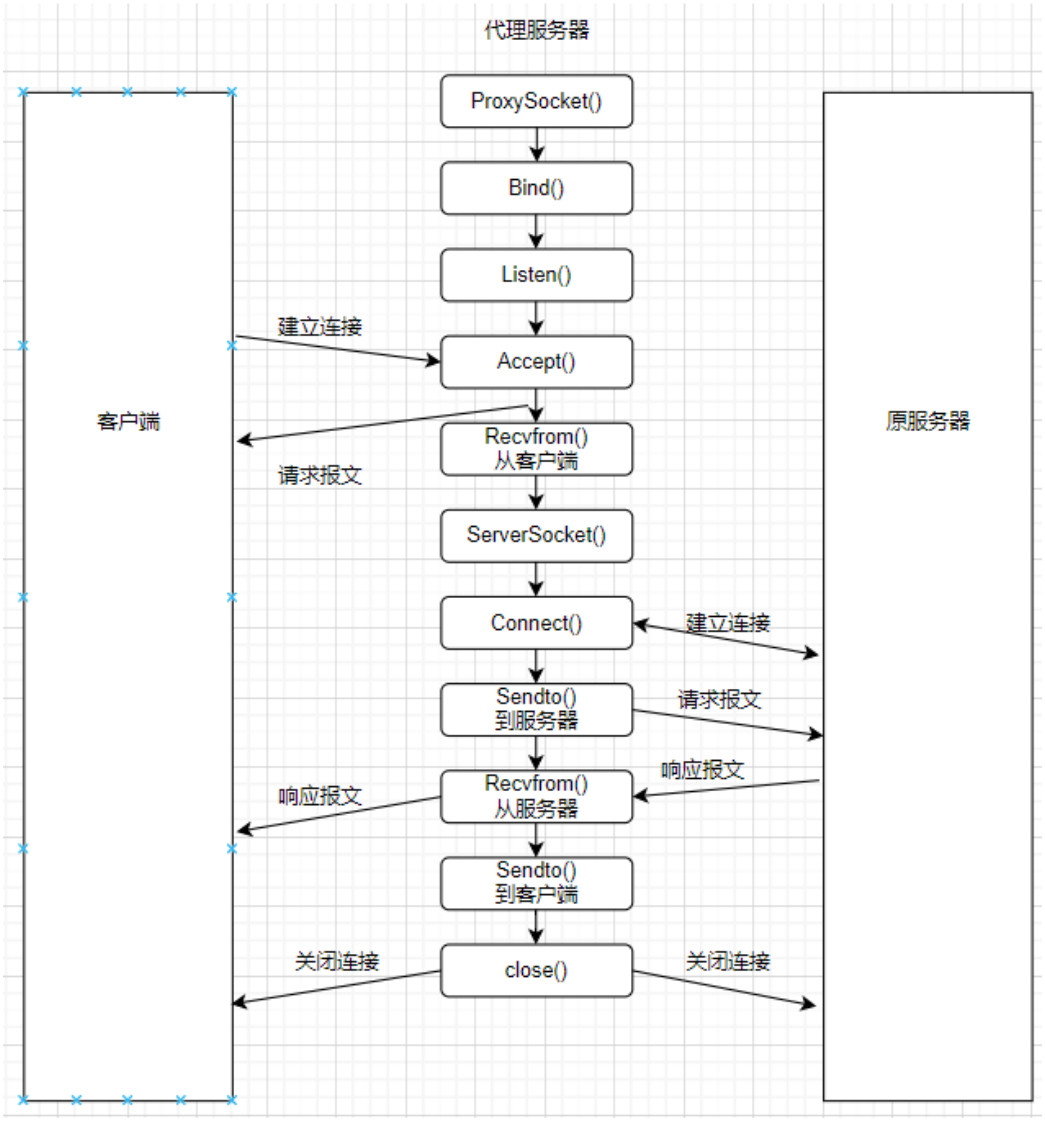


图 3 HTTP 代理服务器的程序流程图

4. 实现 HTTP 代理服务器的关键技术及解决方案

首先我们给出一些前提。定义宏常量，设置我们屏蔽了今日哈工大，钓鱼时从教务处钓鱼到选课系统。我们都是使用HTTP协议而不是HTTPS协议。

```
#define BANNED_WEB "http://today.hit.edu.cn/"         //屏蔽网站
#define PHISHING_WEB_SRC "http://jwc.hit.edu.cn/"     // 钓鱼原网址
#define PHISHING_WEB_DEST "http://jwts.hit.edu.cn/" // 钓鱼目的网址
```

实现基本代理：我们考察示例代码中一开始就给出的四个函数，由此展开：

BOOL InitSocket(); 用于加载套接字库，使用了以下几个函数：

```
socket(AF_INET, SOCK_STREAM, 0);
bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR));
listen(ProxyServer, SOMAXCONN);
```

实现了服务器的socket()、bind()、listen()。初始化一个套接字，bind()将套接字与服务器的host地址绑定，并且绑定端口号。最后用listen()监听。

void ParseHttpHead(char *buffer,HttpHeader * httpHeader); 解析请求报文头，get到报文的method、url、host等。ConnectToServer()将于目标服务器将于目标服务器建立连接。

BOOL ConnectToServer(SOCKET *serverSocket,char *host); 使用socket创建套接字，连接到目标服务器。

unsigned int __stdcall ProxyThread(LPVOID lpParameter); 这里从客户端接受请求报文并发送给服务器，再发送给客户端响应报文。recv()和send()接受客户端的HTTP请求，并通过代理服务器将该请求转发给服务器；服务器将响应发给代理服务器，代理服务器再将响应发送给客户端。

另外在主函数中accept()函数对请求进行接收和响应，对每一个请求代理服务器都是新建一个子线程来接受的；最后处理完成后，关闭线程并清理缓存，然后接收下一个请求。

实现cache功能：将所有的请求的文件保存在本地上。代理服务器第一次与客户端通信后，会留下cache；当客户端再次访问相同的文件时，代理服务器向服务器发送一个请求，该请求需要增加“If-Modified-Since”的头，将服务器发过来的Last-Modified发送回去，给出服务器缓存资源最后修改的时间，服务器通过对比最后修改时间来判断缓存是否过期，如果服务器返回状态码304，表示内容是最新的，代理服务器直接将缓存发送给客户端；如果缓存过期，服务器返回状态码200，目标服务器返回一个新的响应，代理服务器接收后将该响应发回给客户端，并更新本地缓存。

在代码中，我们修改ParseHttpHead()函数解析HTTP头部，判断其中URL是否已经在缓存中。ParseHttpHead()返回Have_cache，可以告知请求页面在代理上是否有缓存，如果有缓存则在客户端请求报文首部插入“If-Modified-Since”的头发送给目标服务器，其返回数据后读取返回的状态及页面最后修改时间。状态码为304或200，按照上面所说的进行操作。如果有错误则跳转到error关闭套接字，结束线程处理。这部分代码范围大，暂时不在此处贴出，后附全部代码。

网站屏蔽：在ProxyThread()中ParseHttpHead()解析TCP报文中的HTTP头部，将报

文头部的host与被屏蔽网站对比，如果一样则直接跳转到error，实现网站屏蔽。

```
// 网站屏蔽
if (strcmp(httpHeader->url, BANNED_WEB) == 0)
{
    printf("网站 %s 已被屏蔽\n", BANNED_WEB);
    goto error;
}
```

用户过滤：例如只允许本机用户访问代理。代码如下：

```
// ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //只允许本机用户访问服务器
```

网站引导：即钓鱼。这部分在ProxyThread()中实现。此部分和网站屏蔽有些相似，也是将请求报文中URL与被引导网站比较，如果一样的话就钓鱼（重定向）到那个引导网站上去。在重定向时我们会将构造的302报文中的地址改为钓鱼的IP，然后将这个改好的302报文send回客户端。网站引导的代码如下：

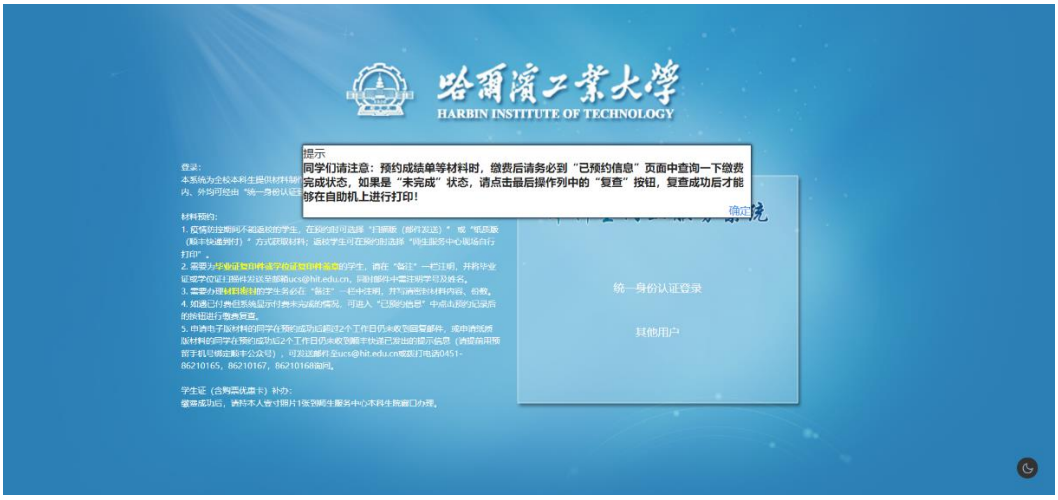
```
//网站钓鱼  访问jwc.hit.edu.cn  重定向到jwts.hit.edu.cn
if (strstr(httpHeader->url, PHISHING_WEB_SRC) != NULL)
{
    char *pr;
    int phishing_len;
    // 打印信息
    printf("网站 %s 已被成功重定向至 %s\n", PHISHING_WEB_SRC, PHISHING_WEB_DEST);
    // 构造报文
    char head1[] = "HTTP/1.1 302 Moved Temporarily\r\n";
    phishing_len = strlen(head1);
    memcpy(phishBuffer, head1, phishing_len);
    pr = phishBuffer + phishing_len;

    char head2[] = "Connection:keep-alive\r\n";
    phishing_len = strlen(head2);
    memcpy(pr, head2, phishing_len);
    pr += phishing_len;

    char head3[] = "Cache-Control:max-age=0\r\n";
    phishing_len = strlen(head3);
    memcpy(pr, head3, phishing_len);
    pr += phishing_len;

    //重定向到jwts.hit.edu.cn
    char phishing_dest[] = "Location: ";
    strcat(phishing_dest, PHISHING_WEB_DEST);
    strcat(phishing_dest, "\r\n\r\n");
    phishing_len = strlen(phishing_dest);
    memcpy(pr, phishing_dest, phishing_len);

    //将302报文返回给客户端
    ret = send(((ProxyParam *)lpParameter)->clientSocket, phishBuffer, sizeof(phishBuffer), 0);
    goto error;
}
```

实验结果：

1. HTTP 代理服务器实验验证过程以及实验结果

1) 首先修改系统代理，之后运行程序。

2) **基本功能**：运行程序后打开一个网站，成功，控制台输出如下。

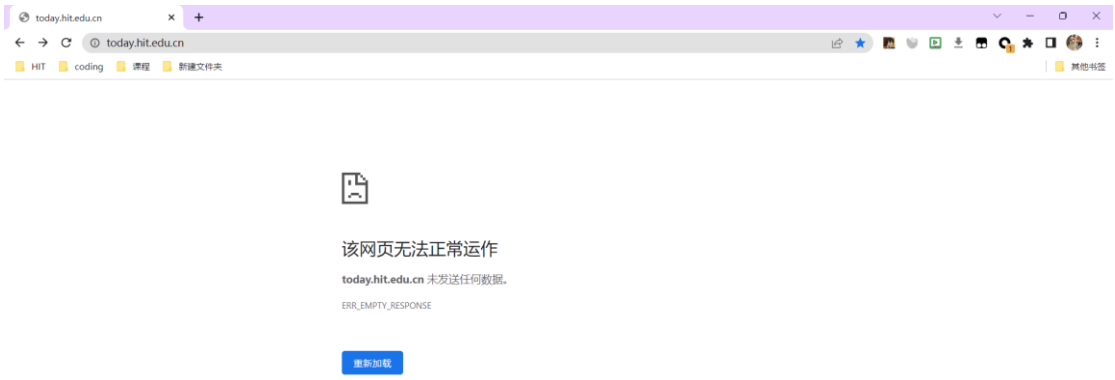


3) **Cache功能：**再次打开jwes网站后可以发现文件夹相应位置有缓存文件。

117.131.23.144	2022/10/7 15:53	144 文件	0 KB
117.184.242.100	2022/10/7 15:53	100 文件	0 KB
117.185.24.177	2022/10/7 15:53	177 文件	0 KB
117.185.24.219	2022/10/7 15:53	219 文件	0 KB
117.185.24.225	2022/10/7 15:53	225 文件	0 KB
117.185.253.224	2022/10/7 15:53	224 文件	0 KB
121.51.73.101	2022/10/7 15:53	101 文件	0 KB
121.51.77.100	2022/10/7 15:53	100 文件	0 KB
175.27.0.201	2022/10/7 15:53	201 文件	0 KB
223.166.152.100	2022/10/7 15:53	100 文件	0 KB
aefd.nelreports.net	2022/10/7 16:39	NET 文件	0 KB
api.adblock-for-youtube.com	2022/10/7 16:37	MS-DOS 应用程序	0 KB
arc.msn.com	2022/10/7 15:26	MS-DOS 应用程序	0 KB
beacons.gcp.gvt2.com	2022/10/7 16:40	MS-DOS 应用程序	0 KB
beacons.gvt2.com	2022/10/7 16:40	MS-DOS 应用程序	0 KB
beacons2.gvt2.com	2022/10/7 16:39	MS-DOS 应用程序	0 KB
beacons3.gvt2.com	2022/10/7 16:39	MS-DOS 应用程序	0 KB
beacons5.gvt3.com	2022/10/7 15:28	MS-DOS 应用程序	0 KB
clientservices.googleapis.com	2022/10/7 16:37	MS-DOS 应用程序	0 KB
cx.icodef.com	2022/10/7 16:37	MS-DOS 应用程序	0 KB
cxcs.microsoft.net	2022/10/7 15:26	NET 文件	0 KB
data.unistream.io	2022/10/7 16:37	IO 文件	0 KB
http	2022/10/7 16:40	文件	0 KB
ids.hit.edu.cn	2022/10/7 15:22	CN 文件	0 KB
login.live.com	2022/10/7 15:22	MS-DOS 应用程序	0 KB
mtalk.google.com	2022/10/7 16:37	MS-DOS 应用程序	0 KB

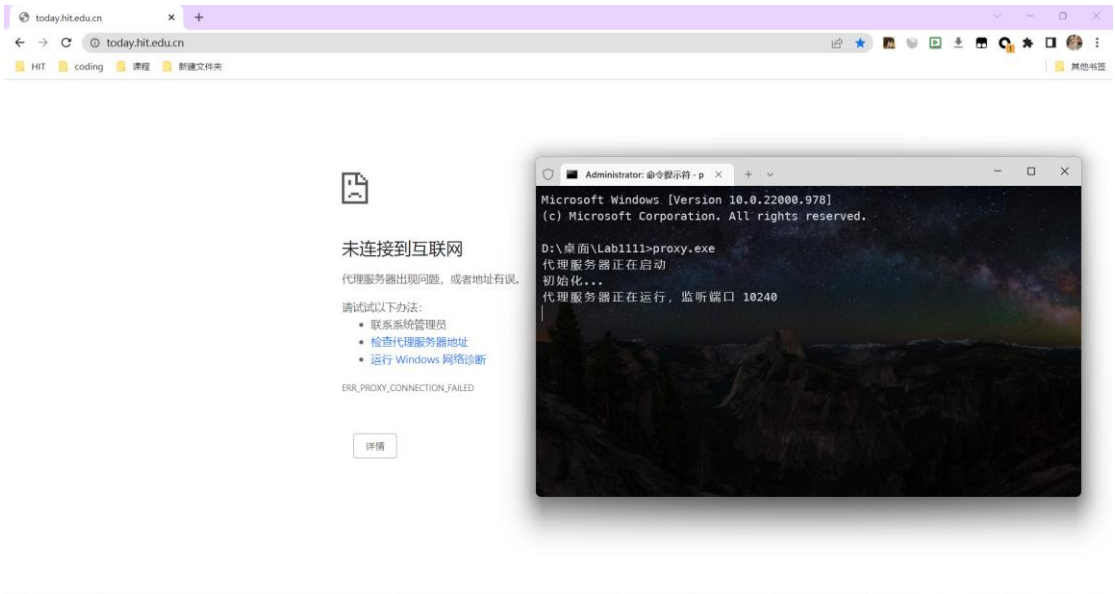
4) **网站过滤：**我们选择屏蔽 http://today.hit.edu.cn ，尝试访问失败，证明网站过滤成功。

```
url: http://today.hit.edu.cn/
代理连接主机 today.hit.edu.cn 成功
网站 http://today.hit.edu.cn/ 已被屏蔽
关闭套接字
```



5) **用户过滤**：修改代码，令本机无访问权限。打开网页无任何响应。

```
// ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.2"); //只允许本机用户访问服务器
```



6) **钓鱼**：我们计划当用户访问 http://jwc.hit.edu.cn 时跳转至 http://jwts.hit.edu.cn 。如下验证。

2.HTTP 代理服务器源代码（带有详细注释）。

见报告后。

问题讨论：

1. 测试缓存时 有时会出现只能接收到200而没有304的情况，这时应该及时主动更换网站；经过测试，换成jwes网站或7k7k小游戏网站、http://info.cern.ch/都是可行的。
2. 一开始我无法使用`#pragma comment(lib,"Ws2_32.lib")`，发现是使用了MinGW编译，而MinGW是不支持上书写法的，这一命令是静态链接Ws2_32.lib库，如果坚持使用MinGW的话就要加上-lws_32。
3. 注意goto语句后不能再定义新变量。
4. 在做网站屏蔽的功能时尽量将请求转到一个html上（可以自己在github上搞一个github page），不然对于某些浏览器可能会反复请求，跳不出循环。
5. 最后在验收时发现很多功能实现都失败了，例如最后的网站过滤、用户过滤等功能都无法正常实现，重启IDE并且最后在助教的帮助下发现是需要另外打开 chrome 的 Disable cache 才能正常运行。感谢陈学长、付学长和杨学姐 >＜!

心得体会：

本次实验让我对TCP协议传输数据的流程和方式有了更深的体会，对 socket 编程有了初步的了解，通过动手实践有了很大收获；掌握了 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更深的认识；同时，也对网站钓鱼、网站屏蔽等有了深刻的理解；感受到了 HTTP cache 的重要作用。

附录：程序源代码

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>
#include <tchar.h>
```

```
#include <fstream>
#include <map>
#include <string>

/*
 * author: 120L020719 zxr
 */

#include <iostream>
using namespace std;

#pragma comment(lib, "Ws2_32.lib")
#define MAXSIZE 65507 * 5 //发送数据报文的最大长度
#define HTTP_PORT 80      // http 服务器端口

#define BANNED_WEB "http://today.hit.edu.cn/"       //屏蔽网站
#define PHISHING_WEB_SRC "http://jwc.hit.edu.cn/"   // 钓鱼原网址
#define PHISHING_WEB_DEST "http://jwts.hit.edu.cn/" // 钓鱼目的网址

// 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));
    }
};

// 结构体 cache
map<string, char *> cache;
struct HttpCache
{
    char url[1024];
    char host[1024];
    char last_modified[200];
    char status[4];
```

```
    char buffer[MAXSIZE];
    HttpCache()
    {
        ZeroMemory(this, sizeof(HttpCache)); // 初始化 cache
    }
};
HttpCache Cache[1024];
int cached_number = 0; //已经缓存的 url 数
int last_cache = 0;    //上一次缓存的索引

BOOL InitSocket();
int ParseHttpHead(char *buffer, HttpHeader *httpHeader);
BOOL ConnectToServer(SOCKET *serverSocket, char *host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void ParseCache(char *buffer, char *status, char *last_modified);

//代理相关参数
SOCKET ProxyServer;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 10240;

// 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 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;
    SOCKADDR_IN acceptAddr;
    ProxyParam *lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;

    //代理服务器不断监听
    while (true)
    {
        acceptSocket = accept(ProxyServer, (SOCKADDR *)&acceptAddr,
NULL);

        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL)
        {
            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 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;
    }
    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); // 创建一个 TCP/IP 协议
族的流套接字
    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"); //只
允许本机用户访问服务器

    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)
{
    char Buffer[MAXSIZE];
    char sendBuffer[MAXSIZE];
    char phishBuffer[MAXSIZE];
    char *CacheBuffer;

    ZeroMemory(Buffer, MAXSIZE);
    ZeroMemory(sendBuffer, MAXSIZE);
    ZeroMemory(phishBuffer, MAXSIZE);

    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    int Have_cache;

    //接收客户端的请求
    recvSize = recv(((ProxyParam *)lpParameter)->clientSocket, Buffer,
MAXSIZE, 0);

    // 为避免报 jump to label 'error'的错误，将其注释
    // if (recvSize <= 0)
    // {
    //  goto error;
    // }
```

```
    HttpHeader *httpHeader = new HttpHeader();
    memcpy(sendBuffer, Buffer, recvSize);
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);
    // ParseHttpHead(CacheBuffer, httpHeader);
    Have_cache = ParseHttpHead(CacheBuffer, httpHeader);
    delete CacheBuffer;

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

    // 网站屏蔽
    if (strcmp(httpHeader->url, BANNED_WEB) == 0)
    {
        printf("网站 %s 已被屏蔽\n", BANNED_WEB);
        goto error;
    }

    //网站钓鱼  访问 jwc.hit.edu.cn  重定向到 jwts.hit.edu.cn
    if (strstr(httpHeader->url, PHISHING_WEB_SRC) != NULL)
    {
        char *pr;
        int phishing_len;
        // 打印信息
        printf("网站 %s 已被成功重定向至 %s\n", PHISHING_WEB_SRC,
PHISHING_WEB_DEST);
        // 构造报文
        char head1[] = "HTTP/1.1 302 Moved Temporarily\r\n";
        phishing_len = strlen(head1);
        memcpy(phishBuffer, head1, phishing_len);
        pr = phishBuffer + phishing_len;

        char head2[] = "Connection:keep-alive\r\n";
        phishing_len = strlen(head2);
        memcpy(pr, head2, phishing_len);
```

```
        pr += phishing_len;

        char head3[] = "Cache-Control:max-age=0\r\n";
        phishing_len = strlen(head3);
        memcpy(pr, head3, phishing_len);
        pr += phishing_len;

        //重定向到 jwts.hit.edu.cn
        char phishing_dest[] = "Location: ";
        strcat(phishing_dest, PHISHING_WEB_DEST);
        strcat(phishing_dest, "\r\n\r\n");
        phishing_len = strlen(phishing_dest);
        memcpy(pr, phishing_dest, phishing_len);

        //将 302 报文返回给客户端
        ret = send(((ProxyParam *)lpParameter)->clientSocket,
phishBuffer, sizeof(phishBuffer), 0);
        goto error;
    }

    //实现 cache 功能
    if (Have_cache) //请求的页面在服务器有缓存
    {
        char cached_buffer[MAXSIZE];
        ZeroMemory(cached_buffer, MAXSIZE);
        memcpy(cached_buffer, Buffer, recvSize);

        //构造缓存的报文头
        char *pr = cached_buffer + recvSize;
        printf(",,");
        memcpy(pr, "If-modified-since: ", 19);
        pr += 19;
        int length = strlen(Cache[last_cache].last_modified);
        memcpy(pr, Cache[last_cache].last_modified, length);
        pr += length;

        //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,
cached_buffer, strlen(cached_buffer) + 1, 0);
        //等待目标服务器返回数据
```

```
        recvSize = recv(((ProxyParam *)lpParameter)->serverSocket,
cached_buffer, MAXSIZE, 0);
        if (recvSize <= 0)
        {
            goto error;
        }

        //解析包含缓存信息的 HTTP 报文头
        CacheBuffer = new char[recvSize + 1];
        ZeroMemory(CacheBuffer, recvSize + 1);
        memcpy(CacheBuffer, cached_buffer, recvSize);

        char last_status[4];    //记录主机返回的状态字
        char last_modified[30]; //记录返回页面的修改时间
        ParseCache(CacheBuffer, last_status, last_modified);

        delete CacheBuffer;

        //分析 cache 的状态字
        if (strcmp(last_status, "304") == 0) // 304 状态码，文件没有被修改
        {
            printf("页面未被修改,缓存 URL:%s\n", Cache[last_cache].url);
            //直接将缓存数据转发给客户端
            ret = send(((ProxyParam *)lpParameter)->clientSocket,
Cache[last_cache].buffer, sizeof(Cache[last_cache].buffer), 0);
            if (ret != SOCKET_ERROR)
                printf("由缓存发送\n");
        }
        else if (strcmp(last_status, "200") == 0) // 200 状态码，表示文件
已被修改
        {
            //首先修改缓存内容
            printf("页面被修改,缓存 URL:%s\n", Cache[last_cache].url);
            memcpy(Cache[last_cache].buffer, cached_buffer,
strlen(cached_buffer));
            memcpy(Cache[last_cache].last_modified, last_modified,
strlen(last_modified));

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

```
            if (ret != SOCKET_ERROR)
                printf("由缓存发送，已修改\n");
        }
    }
    else //没有缓存过该页面
    {
        //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;
        }

        //将该页面缓存到 cache 中

        //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);
        if (ret != SOCKET_ERROR)
        {
            printf("来自服务器\n 成功发送给客户端的报文(目标服务器返回
的)buffer ret = %d \n", ret);
        }
    }
//错误处理
error:
    printf("关闭套接字\n\n");
    Sleep(200);
    closesocket(((ProxyParam *)lpParameter)->clientSocket);
    closesocket(((ProxyParam *)lpParameter)->serverSocket);
    delete lpParameter;
    _endthreadex(0);
    return 0;
}

//************************
// Method: ParseCache
```

```
// FullName: ParseCache
// Access: public
// Returns: void
// Qualifier: 解析 TCP 报文中的 HTTP 头部,在已经 cache 命中的时候使用
// Parameter: char *buffer
// Parameter: char * status
// Parameter: HttpHeader *httpHeader
//************************
void ParseCache(char *buffer, char *status, char *last_modified)
{
    char *p;
    char *ptr;
    const char *delim = "\r\n";
    p = strtok_s(buffer, delim, &ptr); //提取第一行
    memcpy(status, &p[9], 3);
    status[3] = '\0';
    p = strtok_s(NULL, delim, &ptr);
    while (p)
    {
        if (strstr(p, "Last-Modified") != NULL)
        {
            memcpy(last_modified, &p[15], strlen(p) - 15);
            break;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
}

//************************
// Method: ParseHttpHead
// FullName: ParseHttpHead
// Access: public
// Returns: int
// Qualifier: 解析 TCP 报文中的 HTTP 头部
// Parameter: char *buffer
// Parameter: HttpHeader *httpHeader
//************************
int ParseHttpHead(char *buffer, HttpHeader *httpHeader)
{
    int flag = 0; //用于表示 Cache 是否命中，命中为 1，不命中为 0
    char *p;
```

```
    char *ptr;
    const char *delim = "\r\n"; //回车换行符
    p = strtok_s(buffer, delim, &ptr);
    if (p[0] == 'G')
    { // GET 方式
        memcpy(httpHeader->method, "GET", 3);
        memcpy(httpHeader->url, &p[4], strlen(p) - 13);
        printf("url: %s\n", httpHeader->url); // url
        for (int i = 0; i < 1024; i++)
        { //搜索 cache，看当前访问的 url 是否已经存在 cache 中了
            if (strcmp(Cache[i].url, httpHeader->url) == 0)
            { //说明 url 在 cache 中已经存在
                flag = 1;
                break;
            }
        }
        if (!flag && cached_number != 1023) //说明 url 没有在 cache 且 cache
没有满，把这个 url 直接存进去
        {
            memcpy(Cache[cached_number].url, &p[4], strlen(p) - 13);
            last_cache = cached_number;
        }
        else if (!flag && cached_number == 1023) //说明 url 没有在 cache 且
cache 满了,把第一个 cache 覆盖
        {
            memcpy(Cache[0].url, &p[4], strlen(p) - 13);
            last_cache = 0;
        }
    }
    else if (p[0] == 'P') // POST 方式
    {
        memcpy(httpHeader->method, "POST", 4);
        memcpy(httpHeader->url, &p[5], strlen(p) - 14);
        for (int i = 0; i < 1024; i++)
        {
            if (strcmp(Cache[i].url, httpHeader->url) == 0)
            {
                flag = 1;
                break;
            }
        }
```

```
        if (!flag && cached_number != 1023)
        {
            memcpy(Cache[cached_number].url, &p[5], strlen(p) - 14);
            last_cache = cached_number;
        }
        else if (!flag && cached_number == 1023)
        {
            memcpy(Cache[0].url, &p[4], strlen(p) - 13);
            last_cache = 0;
        }
    }

    p = strtok_s(NULL, delim, &ptr);
    while (p)
    {
        switch (p[0])
        {
        case 'H': // HOST
            memcpy(httpHeader->host, &p[6], strlen(p) - 6);
            if (!flag && cached_number != 1023)
            {
                memcpy(Cache[last_cache].host, &p[6], strlen(p) - 6);
                cached_number++;
            }
            else if (!flag && cached_number == 1023)
            {
                memcpy(Cache[last_cache].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;
```

```
            // case '':
        default:
            break;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
    return flag;
}

//***********************************
// 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;
}
```