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	田雪洋		院系	计算机科学与技术学院	
班级	1937102		学号	1190202110	
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：

本次实验的主要目的。

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：

概述本次实验的主要内容，包含的实验项等。

(1) 设计并实现一个基本 HTTP 代理服务器。要求在指定端口（例如 8080）接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址 所指向的 HTTP 服务器（原服务器），接收 HTTP 服务器的响应报文，并 将响应报文转发给对应的客户进行浏览。

(2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓 存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since 头行），向原服务器确认缓存对象是否是最新版本。（选作内容，加分项 目，可以当堂完成或课下完成）

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

a) 网站过滤：允许/不允许访问某些网站；
b) 用户过滤：支持/不支持某些用户访问外部网站；
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站

实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

（1） Socket编程的客户端和服务器端主要步骤：

TCP客户端软件流程

1. 根据目标服务器 IP 地址与端口号
2. 创建套接字，
3. 分配本地端点地址（IP 地址+端口号）
4. 连接服务器：三次握手
5. ，遵循应用层协议进行通信，发送请求报文
6. 接收返回报文，返回 3 或者 5
7. 关闭/释放连接

TCP 服务器端：

主线程 1：创建（主）套接字，并绑定端口号

主线程 2：设置（主）套接字为被动监听模式，准备用于服务器

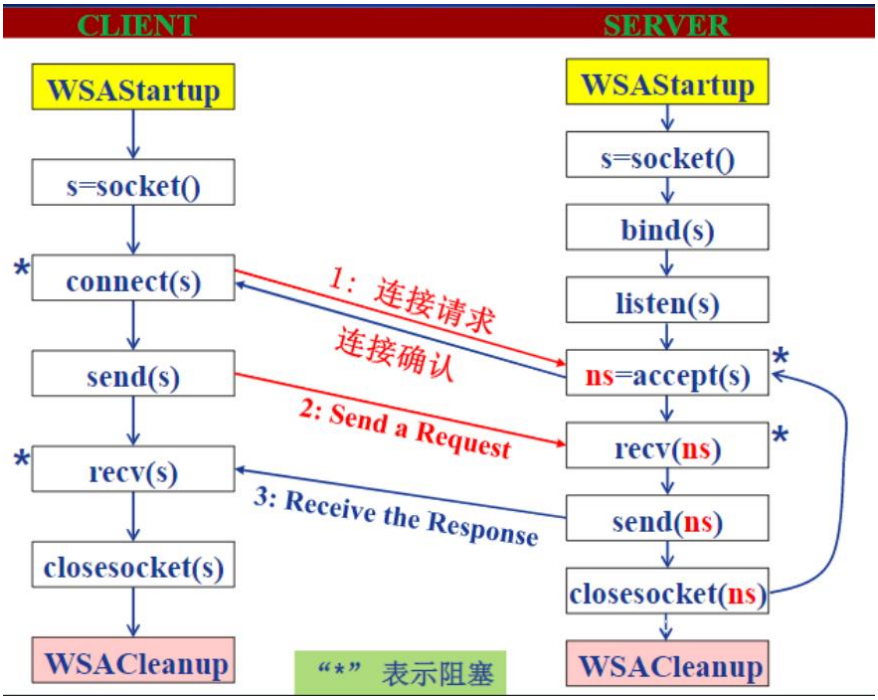
主线程 3：反复调用 accept（）函数接收下一个连接请求（通过主套接字）

子线程 1：接收一个客户的服务请求（通过新创建的套接字）

子线程 2：遵循应用层协议与特定客户进行交互

子线程 3：关闭/释放连接并退出（线程终止）

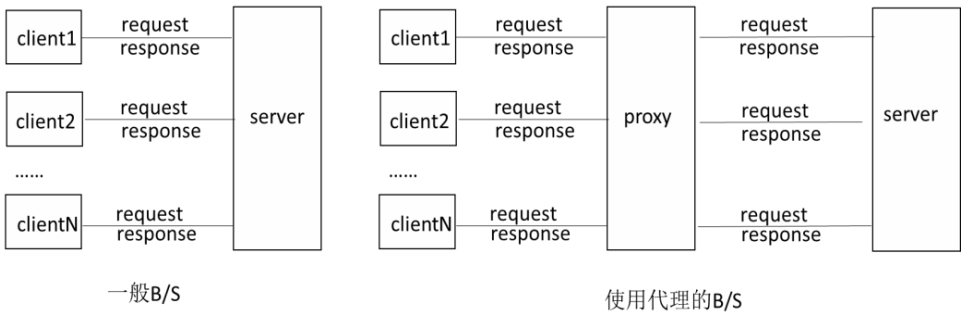
网络应用的 Socket API（TCP）调用基本流程



（2） HHTP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。

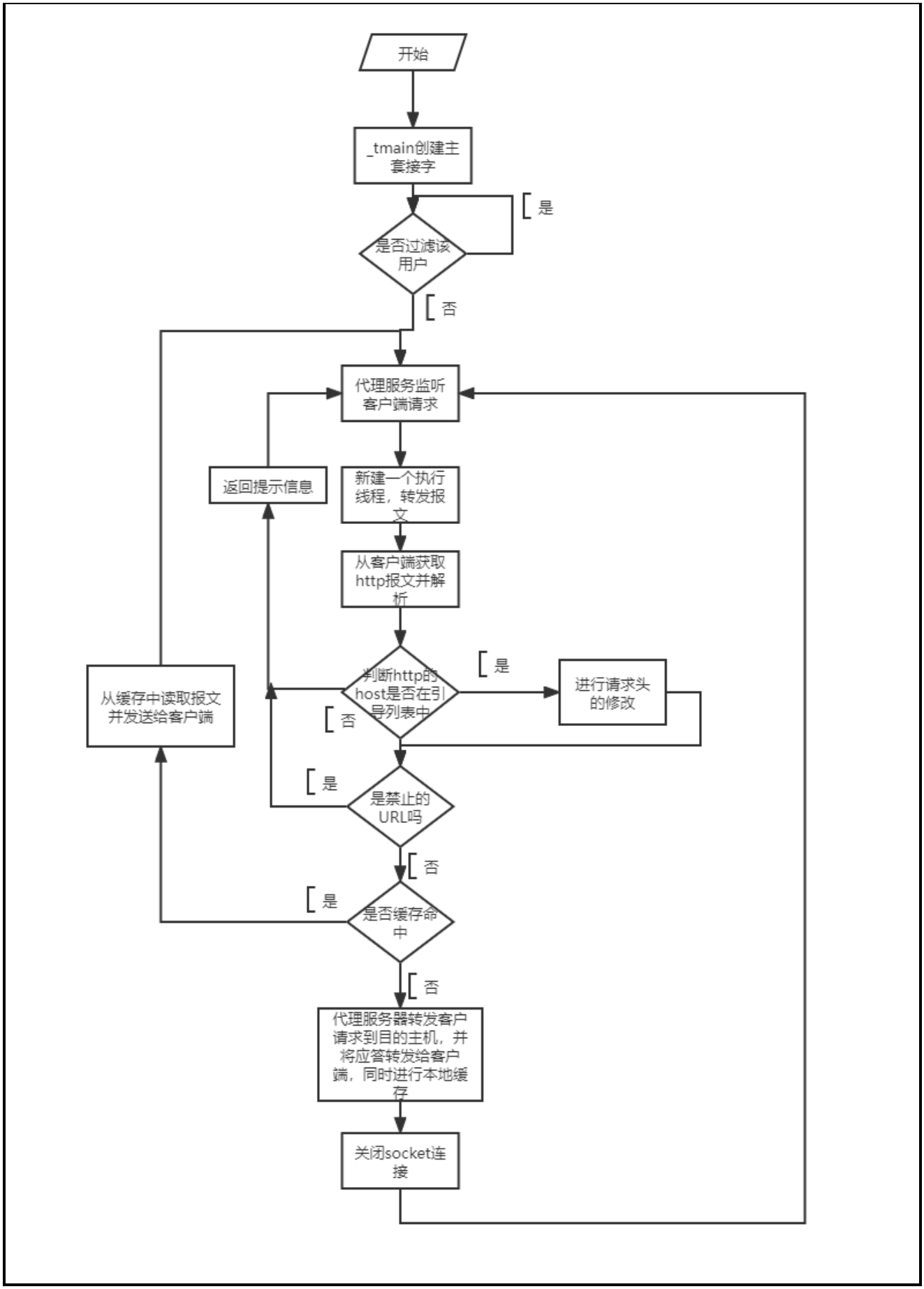
如下为普通 Web 应用通信方式与采用代理服务器的通信方式的对比。



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，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；


开始
_tmain创建主套接字
是否过滤该用户
是
否
代理服务监听客户端请求
新建一个执行线程，转发报文
从客户端获取http报文并解析
判断http的host是否在引导列表中
是
进行请求头的修改
否
是禁止的URL吗
是
返回提示信息
否
是否缓存命中
是
从缓存中读取报文并发送给客户端
否
代理服务器转发客户请求到目的主机，并将应答转发给客户端，同时进行本地缓存
关闭socket连接

（4） 实现HTTP代理服务器的关键技术及解决方案；

1. 关键技术：基本 HTTP 代理服务器的实现
   解决方案：通过老师给出的基本实验框架实现，以下为其中的关键函数和变量

。A）

```
//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 );
    }
};
```

httpHeader 结构体，用于存放报文中的 http 头部数据，其中 method 请求方法，本次实验仅考虑“GET”和“POST”方法。url 为请求访问的网站的 url，host 是目标主机，cookie 存储报文中的 cookie。

B）

```
BOOL InitSocket();//初始化socket
```

作用：初始化代理服务器的 socket，首先加载了套接字库（版本为 2.2），然后设置套接字参数，绑定套接字，进入监听状态。

C）

```
int ParseHttpHead(char* buffer, HttpHeader* httpHeader, char sendBuffer[]);//解析http头部
```

该函数的功能是将从客户端/服务器收到的相应报文通过 buffer，传递进来，并保留一份副本在 sendBuffer 中，然后对 sendbuffer 中的报文逐行解析，获得相应的 http 参数，存放到 httpHeader 中，并且在该函数中完成钓鱼和网站过滤功能（后面介绍）。

D)

```
BOOL ConnectToServer(SOCKET* serverSocket, char* host);//连接到服务器
```

该函数的功能是根据从 host 参数获得的服务器的主机，建立目标服务器的套接字，并连接，返回是否连接成功，连接的套接字通过 serverSocket 参数传回主函数。

E)

```
unsigned int __stdcall ProxyThread(LPVOID lpParameter);//代理服务器线程
```

该函数的作用是为代理服务器服务器创建不同的子线程，在每个子线程中，均完成从客户端获取请求报文，解析该报文，判断本地是否有缓存，向服务器发送请求报文，从服务器接受响应报文/或从本地读取响应报文，向客户端发送相应报文，关闭套接字。该函数传入的参数 LPVOID 是一个没有类型的指针，也就是说你可以将任意类型的指针赋值给 LPVOID 类型的变量（一般作为参数传递），然后在使用的时候在转换回来。

F)

```
int _tmain(int argc, _TCHAR* argv[])
```

主函数，该函数的形式为是 unicode 版本的的 main() 其主要功能就是 main 函数的功能。主要功能为创建主套接字，并绑定端口号，设置（主）套接字为被动监听模式，准备用于服务器，反复调用 accept（）函数接收下一个连接请求（通过主套接字），创建子线程，销毁子线程。

2. 关键技术：cache 功能的实现

解决方案：

首先，我先创建了一个结构体

```
struct HttpCache {
    char url[1024]; // 请求的 url
    char host[1024]; // 目标主机
    string last_modified; //记录上次的修改时间戳
    string status; //状态字
    char buffer[MAXSIZE]; //数据
    HttpCache() {
        ZeroMemory(this, sizeof(HttpCache));
    }
};
```

该结构体是存储缓存的基本单元，各个成员的含义见上面的注释。然后将该结构体存储到一个 vector 容器中，当代理服务器第一次和客户端通信时会保留该页面的信息到缓存，当客户端再次访问该界面时，代理服务器通过判断该页面的 url 和 vector 容器中的 HttpCache 中的 url 是否相等来判断是否为同一个页面。若 Cache 存在，并且需要访问该页面，代理服务器会通过 If-Modified-Since 头将先前目标服务器发送过来的 Last-Modified 时间戳发送回去，让目标服务器验证本地缓存的页面是否是最新的，若不是最新的，则返回 200 和新的内容（该内容会被重新缓存到本地），反之，返回 304 并告诉客户端本地缓存是最新的，然后代理服务器将本地的缓存发送给客户端，具体实现如下：

```
if(!strcmp(httpHeader->method, "GET"))
{
//将客户端发送的请求报文的url和本地缓存的url进行对比，
    //若相等，则缓存命中，然后构造If-modified-since: 头，并将last_modified写入报文
    string str = string(httpHeader->url);
    for ( i = 0; i < cache.size(); ++i)
    {
        if (cache[i].url == str
        )
        printf("\ncache命中\n");
        int len = cache[i].last_modified.size();
        recvSize -= 2;
        memcpy(Buffer+recvSize,"If-modified-since: ",19);
        recvSize += 19;
        memcpy(Buffer+recvSize,cache[i].last_modified.c_str(), cache[i].last_modified.size());
        recvSize += len;
        Buffer[recvSize++] = '\r';
        Buffer[recvSize++] = '\n';
        Buffer[recvSize++] = '\r';
        Buffer[recvSize++] = '\n';
        break;
        }
    }
}
//将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代理服务器从目标服务器收到的信息\n");
printf("%s", Buffer);
//判断目的服务器的发送的报文信息状态字是304还是200
//若为304将本地的缓存发送给客户端
if (!memcmp(&Buffer[9], "304", 3)) {
    ret = send(((ProxyParam*)lpParameter)->clientSocket, cache[i].buffer, sizeof(cache[i].buffer), 0);
    if (ret != SOCKET_ERROR) {
        printf("\n\n**********\n");
        printf("从本地读入缓存，并返回成功");
        printf("\n**********\n\n");
    }
}
//若为200更新本地的缓存，将目的服务器的新报文发送给客户端
else
{
    if (!strcmp(httpHeader->method, "GET") && !memcmp(&Buffer[9], "200", 3))
    {
        char Buffer2[MAXSIZE];
        memcpy(Buffer2, Buffer, sizeof(Buffer));
        const char* delim = "\r\n";
        char* ptr;
        char* p = strtok_s(Buffer2, delim, &ptr);
        bool flag = false;
        //更新时间戳
        while (p) {
            if (strlen(p) >= 15 && !memcmp(p, "Last-Modified: ", 15)) {
```

```
                flag = true;
                break;
            }
            p = strtok_s(NULL, delim, &ptr);
        }
        //添加缓存
        if (flag) {
            HttpCache tempcache;
            memcpy(tempcache.url, httpHeader->url, sizeof(httpHeader->url));
            tempcache.last_modified = p + 15;
            memcpy(tempcache.buffer, Buffer, sizeof(Buffer));
            cache.push_back(tempcache);
        }
    }
    printf("\n\n**********\n");
    printf("本地缓存过时，从目的服务器发送客户端，并更新本地缓存成功");
    printf("\n**********\n\n");
    //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);
}
```

3. 关键技术：网站过滤

   解决方案：

   首先建立了一个 vector 数组用于存储禁止访问的网站

```
//禁止访问的网站表
vector<string>disaccess_website = { "yzb.hit.edu.cn" };
```

   然后，实现了一个函数，通过比较列表中的网站和 http 报文的 host 部分是否相等，来判断是否禁止访问

```
BOOL Unable_access_website(HttpHeader* httpHeader)
{
    string str = httpHeader->host;
    if (find(disaccess_website.begin(), disaccess_website.end(), str) != disaccess_w
    {
        // Element in vector.
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}
```

接着，在解析 http 报文的头部的函数中，将头部信息发送给上述函数进行判断，若禁止访问，则将 http 头部的 url 部分全部置为 0，并打印禁止访问信息

```
//如果httpHeader的host属于禁止访问的网站表
if (Unable_access_website(httpHeader))
{
    printf("\n根据xxx,该网站 %s 已被禁止访问 \n", httpHeader->host);
    memset(httpHeader->host, 0, sizeof(httpHeader->host)); //把需要访问的host全改为0
}
```

4. 关键技术：用户过滤

   解决方案：

   设置一个禁止访问的 ip 地址列表，

```
//禁止访问网站的用户表
vector<string>disaccess_usertable = { "127.0.0.1" };
```

   在_tmain（）函数每次与客户端进行通信时，判断客户端的 IP 地址是否在禁止访问的 ip 地址列表中，若在，则禁止访问，关闭该套接字。

```
if (Unable_access_user(addr_conn))
{
    printf("该用户禁止访问外部网站");
    closesocket(acceptSocket);
    continue;
}
```

5. 关键技术：网站引导

   解决方案：

   设置一个网站引导列表，将网站列表中的第一个网站引导到第二个网站，

```
//钓鱼网站引导表：将用户对前一个网站的访问引导至后一个网站
vector<vector<string>> Phishing_table = { { "http://seie.hit.edu.cn/", "http://cs.hit.edu.cn" } };
```

   在每次进行http头部解析时，将http报文的host部分与网站引导列表中的第一个网站进行比较，若想等，则将http报文的所有host部分全部替换成网站引导列表中的第二个网站。

```
//如果httpHeader的host属于钓鱼网站引导表
if (Phishing_website(httpHeader) != -1)
{
    int i = Phishing_website(httpHeader);
    string target = Phishing_table[i][1];
    target = target.substr(7, target.size() - 1);
    printf("\n%s是钓鱼网站，您即将被钓到其他网站\n", httpHeader->host);
    string str = string(sendBuffer);
    while (str.find(string(httpHeader->host)) != string::npos)  //将发送缓存区的host全部替换为钓鱼网站的host
    {
        str.replace(str.find(string(httpHeader->host)), string(httpHeader->host).size(), target);
    }
    memcpy(sendBuffer, str.c_str(), str.size() + 1); //用新的网站地址替换原buffer_c
    memcpy(httpHeader->host, target.c_str(), target.length() + 1);
}
```

（5） HTTP 代理服务器源代码（带有详细注释）。

```
#include <stdio.h>
#include <iostream>
#include <Windows.h>
#include <winsock.h>
#include <process.h>
#include <string.h>
#include <cstring>
#include <tchar.h>
#include <cstdlib>
#include<vector>
#pragma comment(lib,"Ws2_32.lib")

using namespace std;

#define MAXSIZE 65507 //发送数据报文的最大长度
#define HTTP_PORT 80 //http 服务器端口

//钓鱼网站引导表：将用户对前一个网站的访问引导至后一个网站
vector<vector<string>> Phishing_table = { { "http://seie.hit.edu.cn/", "http://cs.hit.edu.cn" } };

//禁止访问的网站表
vector<string>disaccess_website = { "yzb.hit.edu.cn" };

//禁止访问网站的用户表
vector<string>disaccess_usertable = { "127.0.0.1" };

//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));
```

```
        }
    };
    //缓存的网页
    struct HttpCache {
        char url[1024]; //  请求的  url
        char host[1024]; //  目标主机
        string last_modified; //记录上次的修改时间戳
        string status; //状态字
        char buffer[MAXSIZE]; //数据
        HttpCache() {
            ZeroMemory(this, sizeof(HttpCache));
        }
    };
    //保存缓存网页的数据结构
    vector<HttpCache>cache;

    BOOL InitSocket();//初始化socket
    int ParseHttpHead(char* buffer, HttpHeader* httpHeader, char sendBuffer[]);//解析http头部
    BOOL ConnectToServer(SOCKET* serverSocket, char* host);//连接到服务器
    //LPVOID是一个没有类型的指针，也就是说你可以将任意类型的指针赋值给LPVOID类型的变量（一般作为参数传递），然后在使用的时候在转换回来
    unsigned int __stdcall ProxyThread(LPVOID lpParameter);//代理服务器线程
    BOOL Unable_access_website(HttpHeader* httpHeader);//判断网站是否禁止访问
    int Phishing_website(HttpHeader* httpHeader);//判断该网站是否为钓鱼网站
    BOOL Unable_access_user(SOCKADDR_IN addr_conn);//判断该用户是否能够访问
    int isCache(HttpHeader* httpHeader);
    //代理相关参数
    SOCKET ProxyServer;
    sockaddr_in ProxyServerAddr;
    const int ProxyPort = 12345;//代理服务器的端口

    struct ProxyParam {
        SOCKET clientSocket;
        SOCKET serverSocket;
    };
    //_tmain()不过是unicode版本的的main()._TCHAR表示各个参数，字符串数组的每个单元是char*类型的，指向一个c风格字符串
    //_TCHAR类型是宽字符型字符串，和我们一般常用的字符串不同，它是32位或者更 高的操作系统中所使用的类型.
    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;//先将接收socket置为无效
        ProxyParam* lpProxyParam;
        HANDLE hThread;
        DWORD dwThreadID;

        SOCKADDR_IN addr_conn;
        int nSize = sizeof(addr_conn);
        //通过memset函数初始化内存块
        memset((void*)&addr_conn, 0, sizeof(addr_conn));

        //代理服务器不断监听
        while (true) {
            //返回一个套接字来和客户端通信
            acceptSocket = accept(ProxyServer, NULL, NULL);
            getpeername(acceptSocket, (SOCKADDR*)&addr_conn, &nSize); //获取与addr_conn套接字关联的远程协议地址
            if (Unable_access_user(addr_conn))
            {
                printf("该用户禁止访问外部网站");
                closesocket(acceptSocket);
                continue;
            }
            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 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;
    }
    //创建套接字,SOCK_STREAM面向连接的服务，IPPRPTP_TCP:TCP连接
//SOCK_DGRAM无连接的服务，IPPRPTP_UDP:UDP连接
    ProxyServer = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    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;//使用IPV4地址
    ProxyServerAddr.sin_port = htons(ProxyPort);//绑定端口号
    ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    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];
    ZeroMemory(Buffer, MAXSIZE);
    int i;
    char* CacheBuffer;
    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    HttpHeader* httpHeader = new HttpHeader();
    //接收客户端的请求
    recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);

    printf("请求内容为：\n");
    printf(Buffer);
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);
    ParseHttpHead(CacheBuffer, httpHeader, Buffer); //对请求报文的头部文件进行解析
    delete CacheBuffer;
    if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host)) {    //connect连
接至目标服务器
        goto error;
    }
    printf("代理连接主机  %s成功\n", httpHeader->host);

    if (!strcmp(httpHeader->method, "GET"))
    {
        //将客户端发送的请求报文的url和本地缓存的url进行对比，
        //若相等，则缓存命中，然后构造If-modified-since: 头，并将last_modified写入报文
        string str = string(httpHeader->url);
        for ( i = 0; i < cache.size(); ++i)
        {
            if (cache[i].url == str)
```

```
                {
                printf("\ncache命中\n");
                int len = cache[i].last_modified.size();
                recvSize -= 2;
                memcpy(Buffer+recvSize,"If-modified-since: ",19);
                recvSize += 19;
                memcpy(Buffer+recvSize,cache[i].last_modified.c_str(), cache[i].last_modified.size());
                recvSize += len;
                Buffer[recvSize++] = '\r';
                Buffer[recvSize++] = '\n';
                Buffer[recvSize++] = '\r';
                Buffer[recvSize++] = '\n';
                break;
                }
        }
}
//将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代理服务器从目标服务器收到的信息\n");
printf("%s", Buffer);
//判断目的服务器的发送的报文信息状态字是304还是200
//若为304将本地的缓存发送给客户端
if (!memcmp(&Buffer[9], "304", 3)) {
        ret = send(((ProxyParam*)lpParameter)->clientSocket, cache[i].buffer, sizeof(cache[i].buffer), 0);
        if (ret != SOCKET_ERROR) {
                printf("\n\n**************************\n");
                printf("从本地读入缓存，并返回成功");
                printf("\n**************************\n\n");
        }
}
//若为200更新本地的缓存，将目的服务器的新报文发送给客户端
else
{
        if (!strcmp(httpHeader->method, "GET") && !memcmp(&Buffer[9], "200", 3))
        {
                char Buffer2[MAXSIZE];
                memcpy(Buffer2, Buffer, sizeof(Buffer));
                const char* delim = "\r\n";
                char* ptr;
```

```
                    char* p = strtok_s(Buffer2, delim, &ptr);
                    bool flag = false;
                    //更新时间戳
                    while (p) {
                        if (strlen(p) >= 15 && !memcmp(p, "Last-Modified: ", 15)) {
                            flag = true;
                            break;
                        }
                        p = strtok_s(NULL, delim, &ptr);
                    }
                    //添加缓存
                    if (flag) {

                        HttpCache tempcache;
                        memcpy(tempcache.url, httpHeader->url, sizeof(httpHeader->url));
                        tempcache.last_modified = p + 15;
                        memcpy(tempcache.buffer, Buffer, sizeof(Buffer));
                        cache.push_back(tempcache);
                    }
                }

    printf("\n\n*************************************************************************\n");
                printf("本地缓存过时或者第一次访问该页面，从目的服务器发送客户端，并更新本地缓存成
功");

    printf("\n*************************************************************************\n\n");
                //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
//*************************************
int ParseHttpHead(char* buffer, HttpHeader* httpHeader, char sendBuffer[]) {
	char* p;
	char* ptr;
	const char* delim = "\r\n"; //回车换行符
	int flag = -1; //-1表示没有缓存，其余数字表示在缓存数组里的位置下标
	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);
		printf("url：%s\n", httpHeader->url);
	}
	else if (p[0] == 'P') {      //POST方式
		memcpy(httpHeader->method, "POST", 4);
		memcpy(httpHeader->url, &p[5], strlen(p) - 14);
	}

	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);
	}
```

```
        //如果httpHeader的host属于禁止访问的网站表
        if (Unable_access_website(httpHeader))
        {
            printf("\n根据xxx,该网站 %s 已被禁止访问 \n", httpHeader->host);
            memset(httpHeader->host, 0, sizeof(httpHeader->host)); //把需要访问的host全改为0
        }
        //如果httpHeader的host属于钓鱼网站引导表
        if (Phishing_website(httpHeader) != -1)
        {
            int i = Phishing_website(httpHeader);
            string target = Phishing_table[i][1];
            target = target.substr(7, target.size() - 1);
            printf("\n%s是钓鱼网站，您即将被钓到其他网站\n", httpHeader->host);
            string str = string(sendBuffer);
            while (str.find(string(httpHeader->host)) != string::npos)   //将发送缓存区的host全部替换为钓鱼网站的host
            {
                str.replace(str.find(string(httpHeader->host)), string(httpHeader->host).size(), target);
            }
            memcpy(sendBuffer, str.c_str(), str.size() + 1); //用新的网站地址替换原buffer_c
            memcpy(httpHeader->host, target.c_str(), target.length() + 1);
        }
        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;
        }
        //printf(host);
        in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
```

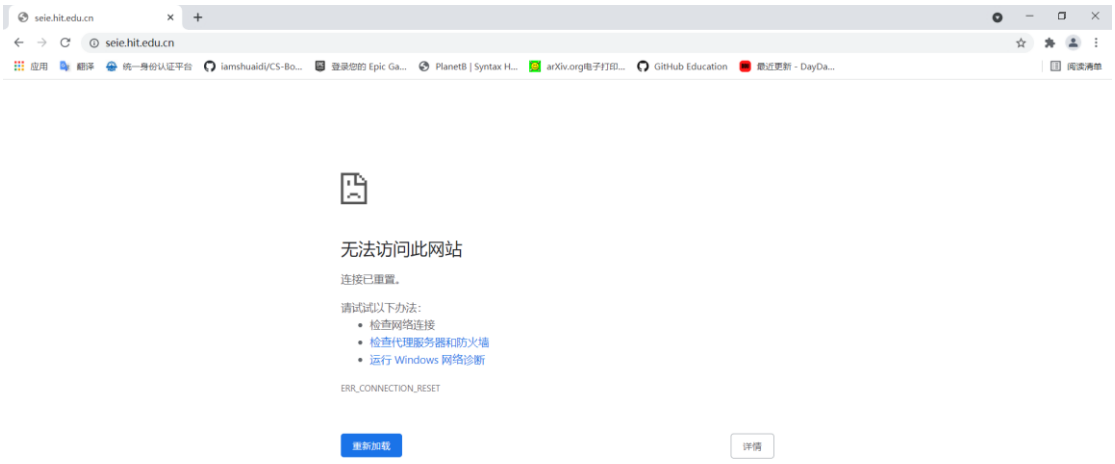
```
        serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
        *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (*serverSocket == INVALID_SOCKET) {
            return FALSE;
        }
        if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
            closesocket(*serverSocket);
            return FALSE;
        }
        return TRUE;
    }
    //*************************************
    // Method:  网站过滤
    // FullName: Unable_access_website
    // Access: public
    // Returns: BOOL
    // Qualifier:  根据域名是否在禁止名单中
    // Parameter: HttpHeader *httpHeader
    //*************************************
    BOOL Unable_access_website(HttpHeader* httpHeader)
    {
        string str = httpHeader->host;
        if (find(disaccess_website.begin(), disaccess_website.end(), str) != disaccess_website.end())
        {
            // Element in vector.
            return TRUE;
        }
        else
        {
            return FALSE;
        }
    }
    //*************************************
    // Method:  钓鱼网站
    // FullName:Phishing_website
    // Access: public
    // Returns: BOOL
    // Qualifier:  根据解析出来的http的url是否在
    //  钓鱼网站的列表中
    // Parameter: HttpHeader *httpHeader
    //*************************************
    int Phishing_website(HttpHeader* httpHeader)
    {
        string str = httpHeader->url;
```

```
        int n = Phishing_table.size();
        for (int i = 0; i < n; ++i)
        {
            if (str == Phishing_table[i][0])
            {
                // Element in vector.
                return i;
            }
        }
        return -1;
    }
    //************************************
    // Method:  用户过滤
    // FullName: Unable_access_user
    // Access: public
    // Returns: BOOL
    // Qualifier:  根据域名是否在禁止名单中
    // Parameter: SOCKADDR_IN addr_conn
    //************************************
    BOOL Unable_access_user(SOCKADDR_IN addr_conn)
    {
        char* p = inet_ntoa(addr_conn.sin_addr);
        string str = string(p);
        if (find(disaccess_usertable.begin(), disaccess_usertable.end(), str) != disaccess_usertable.end())
        {
            // Element in vector.
            return TRUE;
        }
        else
        {
            return FALSE;
        }
    }
```

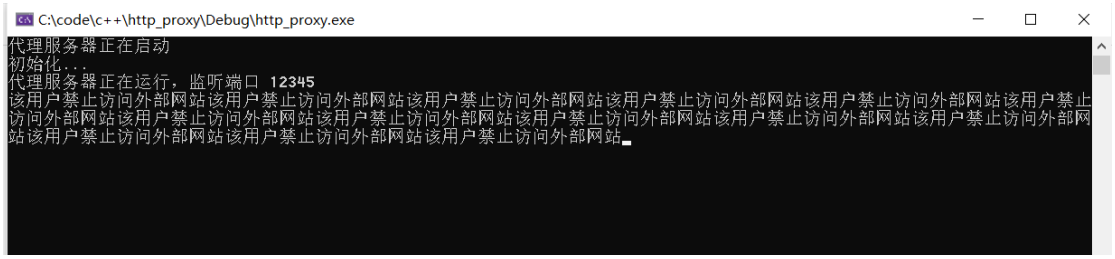
实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

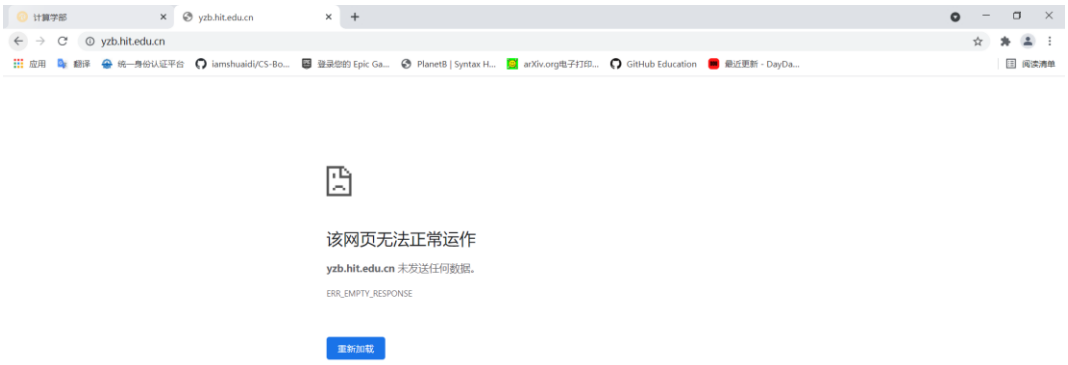
1. 禁止用户访问：设置本地回环地址127.0.0.1为禁止访问地址，在Chrome浏览器进行测试，测试结果如下：



在控制台打印如下：



2. 网站过滤：设置yzb.hit.edu.cn为禁止访问，测试结果如下：



```
请求内容为：
GET http://yzb.hit.edu.cn/ HTTP/1.1
Host: yzb.hit.edu.cn
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/
537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sig
ned-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9

url: http://yzb.hit.edu.cn/

根据xxx,该网站 yzb.hit.edu.cn 已被禁止访问
```

3. 网站引导：将seie.hit.edu.cn引导到cs.hit.edu.cn，测试结果如下：


```
请求内容为：
GET http://seie.hit.edu.cn/ HTTP/1.1
Host: seie.hit.edu.cn
Proxy-Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/
537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sig
ned-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=854438EC06791D2137EBB8B9902B441D

url: http://seie.hit.edu.cn/

seie.hit.edu.cn是钓鱼网站，您即将被钓到其他网站
代理连接主机 cs.hit.edu.cn成功
```

4. Cache缓存：

初次访问结果如下：

```
请求内容为：
GET http://hituc.hit.edu.cn/_upload/article/images/38/5c/a3144d9c4f2eb8294104c22c9566/db4f9915-bfff-4bf1-b039-5ca1fc2a2e
ec.jpg HTTP/1.1
Host: hituc.hit.edu.cn
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/
537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://hituc.hit.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=AB83757AA8BAD92C0DE7BBEAB3FCDBA5

url: http://hituc.hit.edu.cn/_upload/article/images/38/5c/a3144d9c4f2eb8294104c22c9566/db4f9915-bfff-4bf1-b039-5ca1fc2a2
eec.jpg
代理连接主机 hituc.hit.edu.cn成功
```

```
代理服务器从目标服务器收到的信息
HTTP/1.1 200 200
Date: Sat, 30 Oct 2021 02:58:00 GMT
Server: Server
X-Frame-Options: SAMEORIGIN
Frame-Options: SAMEORIGIN
X-Application-Context: application
Set-Cookie: JSESSIONID=AB83757AA8BAD92C0DE7BBEAB3FCDBA5; Path=/; HttpOnly
Content-Length: 0
Connection: close

if,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://hituc.hit.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9



**********
本地缓存过时或者第一次访问该页面，从目的服务器发送客户端，并更新本地缓存成功
**********
```

再次访问结果如下：

```
cache命中

代理服务器从目标服务器收到的信息
HTTP/1.1 304 Not Modified
Date: Sat, 30 Oct 2021 03:00:15 GMT
Server: Server
X-Frame-Options: SAMEORIGIN
Frame-Options: SAMEORIGIN
Last-Modified: Wed, 07 Jul 2021 06:38:15 GMT
ETag: "29132-5c682c8821c82"
Accept-Ranges: bytes
Connection: close

t/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://hituc.hit.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=AB83757AA8BAD92C0DE7BBEAB3FCDBA5
If-modified-since: Wed, 07 Jul 2021 06:38:15 GMT



**********
从本地读入缓存，并返回成功
**********
```

问题讨论：

问题1.不会使用socket编程

解决方案：通过mooc和c语言中文网自学。

问题2对http报文结构不了解

解决方案：通过相关文档和博客自学。

心得体会：

结合实验过程和结果给出实验的体会和收获。

纸上得来终觉浅，绝知此事要躬行。通过本次实验更加深刻地理解了http传送过程和socket编程。本次实验设计不难，但是在实现上有不小的难度，可能是因为我c++编程和http理解不够，但是通过本次实验，很好地加深对这些的理解。