哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

<table>
<tr><td>实验名称</td><td colspan="5">HTTP 代理服务器的设计与实现</td></tr>
<tr><td>姓名</td><td colspan="2">余涛</td><td>院系</td><td colspan="2">计算机科学与技术学院</td></tr>
<tr><td>班级</td><td colspan="2">1803202</td><td>学号</td><td colspan="2">1180300829</td></tr>
<tr><td>任课教师</td><td colspan="2">刘亚维</td><td>指导教师</td><td colspan="2">刘亚维</td></tr>
<tr><td>实验地点</td><td colspan="2">格物 207</td><td>实验时间</td><td colspan="2">2020.10.31</td></tr>
<tr><td rowspan="2">实验课表现</td><td>出勤、表现得分(10)</td><td></td><td rowspan="2">实验报告
得分(40)</td><td rowspan="2"></td><td rowspan="2">实验总分</td><td rowspan="2"></td></tr>
<tr><td>操作结果得分(50)</td><td></td></tr>
<tr><td colspan="6">教师评语</td></tr>
<tr><td colspan="6"></td></tr>
</table>

计算学部

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（钓鱼）。

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：

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），返回 3 或者 5
5. 关闭连接（closesocket）

TCP 服务器端：

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）返回 3 或者 5
5. 关闭连接（closesocket）返回2

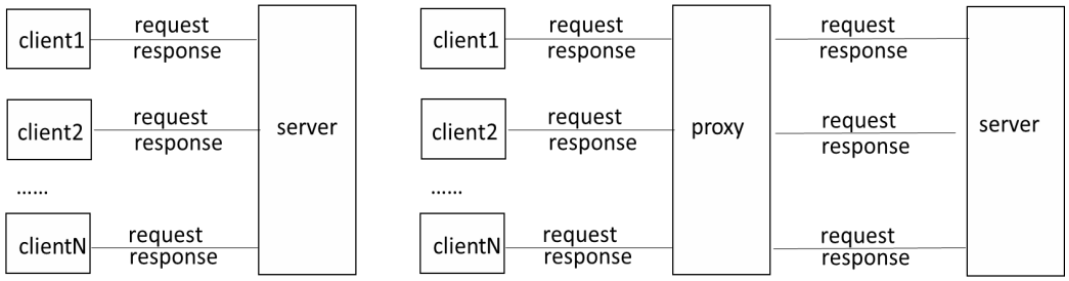
(2) HTTP 代理服务器的基本原理

HTTP 代理服务器的主要功能：

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。如图所示，为普通 Web 应用通信方式与

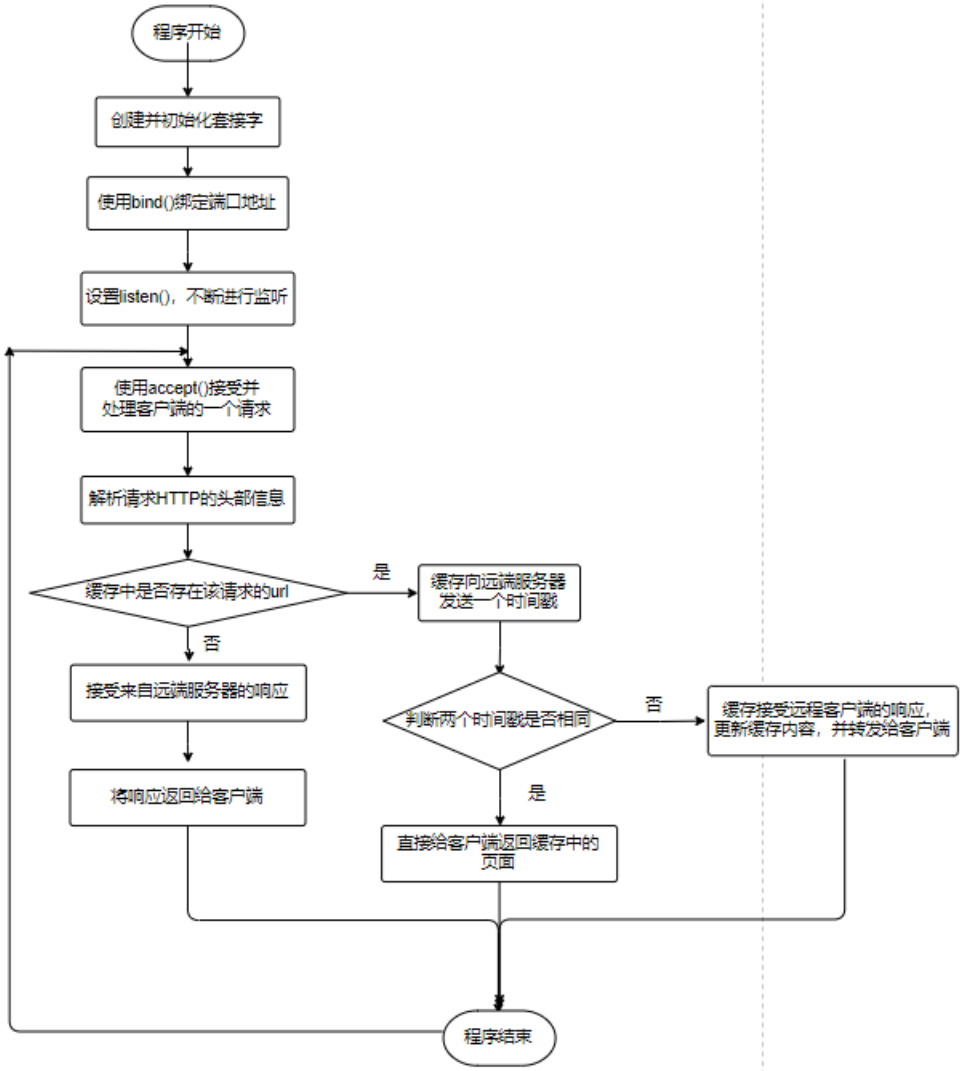
采用代理服务器的通信方式的对比。



具体实现原理：

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）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：通过老师给定参考代码的几个函数来实现

(a) BOOL InitSocket()

作用：创建并初始化套接字，加载套接字库，绑定端口地址。

实现：首先加载套接字库，然后定义版本为 2.2，加载 dll 文件的套接字库，对于各种加载错误打印错误提示。

函数中使用以下几个 socket 函数：

WSAStartup(WORD wVersionRequested,LPWSADATA lpWSAData)

socket(AF_INET, SOCK_STREAM, 0);

bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR))；和

listen(ProxyServer, SOMAXCONN)

InitSocket 实现了服务器流程中的 socket 和 bind 和 listen；

(b) BOOL ParseHttpHead(char *buffer, HttpHeader * httpHeader, char sendBuffer[])

作用：对请求的 TCP 报文的 HTTP 头部文件进行解析，得到请求报文中的 method, url, host 和 cookie 等，然后用于 ConnectToServer 函数与目标服务器建立连接。

实现：由于实现了缓存功能，所以需要对老师的代码进行一些功能的增加。对于 GET 和 POST

两种方式中都增加了对于 cache 缓存遍历的功能，然后对于 HttpHead 的 host 属于禁止访问的网站表的进行相应处理，对于 HttpHead 的 host 属于钓鱼网站引导表的进行相应的处理。

(3) BOOL ConnectToServer(SOCKET *serverSocket, char *host)

作用：根据主机创建目标服务器套接字，并连接使用 socket 创建套接字，connect 连接至目标服务器

实现：创建服务器套接字并连接即可。

(4) unsigned int __stdcall ProxyThread(LPVOID lpParameter)

作用：线程执行函数，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。

实现：首先通过ParseHttpHead函数基对请求报文头部进行解析，然后将得到的HTTP头部文件用作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。由于实现了缓存功能，所以需要对老师的代码进行一些功能的增加。对于缓存命中的情况下，需要构建一个用于缓存的请求报文头，将客户端发送的HTTP数据报文直接转发给目标服务器，等待服务器返回数据，解析包含缓存信息的HTTP报文头，通过分析cache的状态码来判断页面是否被修改，若状态码为200，则说明页面被修改，需要服务器将最新的数据发送给缓存，然后缓存保存并转发给客服端。若状态码为304，说明页面没有被修改，直接将缓存中的数据转发给客户端即可。

2. 关键技术：Cache功能的实现

解决方案：

首先定义一个HttpCache的结构体：

```
//cache缓存 存储数据结构
map<string, char*>cache;
struct HttpCache {
    char url[1024];  //储存的url
    char host[1024]; //目标主机
    char last_modified[200]; //记录上次的修改时间戳
    char status[4]; //状态字
    char buffer[MAXSIZE]; //数据
    HttpCache() {
        ZeroMemory(this, sizeof(HttpCache));
    }
};
```

然后定义一个大小为1024的Cache数组

```
HttpCache Cache[1024];
int cached_quantities = 0;//初始化已经缓存的url数
int last_cache_location = 0;//初始化上一次缓存的索引
```

当代理服务器第一次和客户端通信时会保留该页面到Cache中，当客户端再次发送同样页面的请求时，需要首先判断Cache中是否已经有此页面，若有则说明缓存命中。只需要通过遍历Cache即可，一次搜素缓存Cache，若当前请求的url存在Cache中则说明缓存命中，直接退出遍历；若当前url没有存在Cache中，且Cache还存在空闲空间，则将该页面存入Cache中，若当前url不在Cache中，且Cache已经满了，则用该页面覆盖掉Cache中的第一个页面，具体实现方法为：

```
for (int i = 0; i < 1024; i++) { //依次搜索缓存cache，确定当前访问的url是否已经存在cache中
    if (strcmp(Cache[i].url, httpHeader->url) == 0) { //当前url在已经存在cache中
        flag = 1;
        break;
    }
}
if (!flag && cached_quantities != 1023) {//当前url没有存在cache中，cache还存在空闲空间，往cache中存入url
    memcpy(Cache[cached_quantities].url, &p[4], strlen(p) - 13);
    last_cache_location = cached_quantities;
}
else if (!flag && cached_quantities == 1023) {//当前url没有存在cache中，但是cache已满,用该url覆盖第一个cache
    memcpy(Cache[0].url, &p[4], strlen(p) - 13);
    last_cache_location = 0;
}
```

若Cache存在需要访问的页面时，代理服务器会通过 If-Modified-Since 头将先前目标服务器端发过来的 Last-Modified 最后修改时间戳发送回去，让目标服务器端进行验证，通过这个时间戳判断客户端的页面是否是最新的，如果不是最新的，则返回200和新的内容，如果是最新的，则返回 304 并告诉客户端其本地Cache的页面是最新的，于是代理服务器将本地Cache的页面直接发送给客户端即可，具体实现为：

```
//构造一个用于缓存的请求报文头
char* pr = cached_buffer + recvSize;
memcpy(pr, "If-modified-since: ", 19);  //标准的HTTP请求头标签
pr += 19;
int lenth = strlen(Cache[last_cache_location].last_modified);
memcpy(pr, Cache[last_cache_location].last_modified, lenth);
pr += lenth;
```

```
//分析cache的状态码
if (strcmp(last_status, "304") == 0) {//如果页面没有被修改，状态码为304
    printf("\n页面没有修改过\n缓存的url为:%s\n", Cache[last_cache_location].url);
    //将缓存的数据直接转发给客户端
    ret = send(((ProxyParam*)lpParameter)->clientSocket, Cache[last_cache_location].buffer, sizeof(Cache[last_cache_location].buffer), 0);
    if (ret != SOCKET_ERROR)
    {
        printf("页面来自未修改过的缓存\n");
    }
}
else if (strcmp(last_status, "200") == 0) {//如果页面已经已经修改了缓存中的内容，状态码为200
    printf("\n页面被修改过\n缓存的url为:%s\n", Cache[last_cache_location].url);
    memcpy(Cache[last_cache_location].buffer, cached_buffer, strlen(cached_buffer));
    memcpy(Cache[last_cache_location].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("页面来自修改过的缓存\n");
    }
}
```

3. 关键技术：网站过滤：

解决方案：

定义一个禁止访问的网站表：

```
//禁止访问的网站表
set<string> No_access_web_table =
{
    "www.enshi.gov.cn",
    //"www.badong.net",
};
```

对于ParseHttpHead解析TCP报文中的HTTP头部，遍历禁止访问网站表，将请求报文头部中的host与禁止访问网站表中的每一个网站进行比较，如果出现相同的表示访问的网站被禁止访问，将该网站的host改为全0，具体实现为：

```
//如果httpHeader的host属于禁止访问的网站表
if (No_access_web_table.find(string(httpHeader->host)) != No_access_web_table.end())
{
    printf("该网站 %s 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：

定义一个禁止访问网站的用户表：

```
//禁止访问网站的用户表
set<string> No_access_user_table =
{
    "127.0.0.0"
};
```

在主函数中，对于建立起客户端和代理服务器的连接每次连接，得到客户端的ip地址，遍历禁止访问网站表。将客户端的ip地址与禁止访问网站的用户表中的每一个ip比较，如果相同，则跳过此次监听，具体实现为：

```
//禁止访问网站的用户跳过本次循环，执行下一次监听
if (No_access_user_table.find(string(inet_ntoa(addr_conn.sin_addr))) != No_access_user_table.end())
{
    printf("用户 %s没有权限，禁止访问该网站 \n", inet_ntoa(addr_conn.sin_addr));
    continue;
}
```

5. 关键技术：网站引导：

解决方案：

定义一个钓鱼网站引导表：

```
//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
map<string, string> Fishing_site_guide_table =
{
    { "hitgs.hit.edu.cn", "today.hit.edu.cn" },
    { "", "" }
};
```

类似于网站过滤，遍历禁止访问网站表，如果请求报文头部中的url与禁止访问网站表中的每一个跳转前网站的url相同，就将该url改为跳转后的网站的url，具体实现为：

```
//如果httpHeader的host属于钓鱼网站引导表
else if (Fishing_site_guide_table.find(string(httpHeader->host)) != Fishing_site_guide_table.end())
{
    printf("引导至钓鱼网站 %s 成功\n", httpHeader->host);
    string target = Fishing_site_guide_table[string(httpHeader->host)];
    const char* target_c = target.c_str();
    replace(sendBuffer, string(httpHeader->host), target);  //用后一个host代替前一个host
    memcpy(httpHeader->host, target_c, target.length() + 1);
}
return flag;
```

(5) HTTP 代理服务器实验验证过程以及实验结果

实验验证方法：

1. 基本HTTP代理服务器的实现：

为自己的浏览器设置一个ip地址为127.0.0.1，端口号为10240的代理。
然后运行程序，看能否正常访问http://today.hit.edu.cn/，观察打印请求
2. Cache功能的实现：
通过多次访问同一个网站http://www.badong.net/，观察打印请求是否返回304 Not Modified
3. 网站过滤：
访问禁止访问网站表中的网站http://www.enshi.gov.cn/，观察打印请求
4. 用户过滤：
将禁止访问网站的用户表中一个用户ip设置为：127.0.0.1（本机），然后访问任意一个网站，观察打印请求
5. 网站引导：
访问钓鱼网站引导表中的http://hitgs.hit.edu.cn/，看是否引导至http://today.hit.edu.cn/，观察打印请求

（6）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 <map>
#include <cstdlib>
#include <set>
#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
map<string, string> Fishing_site_guide_table =
{
    { "hitgs.hit.edu.cn", "today.hit.edu.cn" },
    { "", "" }
};

//禁止访问的网站表
set<string> No_access_web_table =
{
    "www.enshi.gov.cn",
    //"www.badong.net",
};
```

```
//禁止访问网站的用户表
set<string> No_access_user_table =
{
    "127.0.0.0"
};


//cache缓存 存储数据结构
map<string, char*>cache;
struct HttpCache {
    char url[1024];    //储存的url
    char host[1024]; //目标主机
    char last_modified[200]; //记录上次的修改时间戳
    char status[4]; //状态字
    char buffer[MAXSIZE]; //数据
    HttpCache() {
        ZeroMemory(this, sizeof(HttpCache));
    }
};
HttpCache Cache[1024];
int cached_quantities = 0;//初始化已经缓存的url数
int last_cache_location = 0;//初始化上一次缓存的索引

//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));
    }
};

BOOL InitSocket();
int ParseHttpHead(char* buffer, HttpHeader* httpHeader, char sendBuffer[]);
BOOL ConnectToServer(SOCKET* serverSocket, char* host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void ParseCacheHead(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 _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;

    SOCKET com_Sock;
    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) {
        acceptSocket = accept(ProxyServer, NULL, NULL);
        com_Sock = acceptSocket;
        getpeername(com_Sock, (SOCKADDR*)& addr_conn, &nSize); //获取与
addr_conn套接字关联的远程协议地址

        //禁止访问网站的用户跳过本次循环，执行下一次监听
        if (No_access_user_table.find(string(inet_ntoa(addr_conn.sin_addr))) !=
No_access_user_table.end())
        {
            printf("用户  %s没有权限，禁止访问该网站  \n",
```

```
inet_ntoa(addr_conn.sin_addr));
                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;
        }
        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;
        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);

        //char sendBuffer[MAXSIZE];
        //ZeroMemory(sendBuffer, MAXSIZE);


        char* CacheBuffer;
        SOCKADDR_IN clientAddr;
        int length = sizeof(SOCKADDR_IN);
        int recvSize;
        int ret;
```

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

        //cache缓存定义变量
        int whether_exist_cache;
        char* cacheBuffer0 = new char[MAXSIZE];
        char* p;
        map<string, char*>::iterator iter;
        string sp;

        //接收客户端的请求
        recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
        if (recvSize <= 0) {
            goto error;
        }
        printf("请求内容为：\n");
        printf(Buffer);
        //memcpy(sendBuffer, Buffer, recvSize);

        CacheBuffer = new char[recvSize + 1];
        ZeroMemory(CacheBuffer, recvSize + 1);
        memcpy(CacheBuffer, Buffer, recvSize);
        whether_exist_cache = ParseHttpHead(CacheBuffer, httpHeader, Buffer); //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等，返回url是否存在于缓存中，用于ConnectToServer函数与目标服务器建立连接
        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);

        //对于请求有缓存的情况下
        if (whether_exist_cache)
        {
            char cached_buffer[MAXSIZE];
            ZeroMemory(cached_buffer, MAXSIZE);
            memcpy(cached_buffer, Buffer, recvSize);

            //构造一个用于缓存的请求报文头
            char* pr = cached_buffer + recvSize;
            memcpy(pr, "If-modified-since: ", 19);   //标准的HTTP请求头标签
            pr += 19;
            int lenth = strlen(Cache[last_cache_location].last_modified);
            memcpy(pr, Cache[last_cache_location].last_modified, lenth);
```

```
                pr += lenth;


                //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]; //用于记录服务器主机返回的状态码(包括304和200)
                char last_modified[30];//用于记录记住返回的页面修改的时间
                ParseCacheHead(CacheBuffer, last_status, last_modified);
                delete CacheBuffer;

                //分析cache的状态码
                if (strcmp(last_status, "304") == 0) {//如果页面没有被修改，状态码为304
                    printf("\n页面没有修改过\n缓存的url为:%s\n",
Cache[last_cache_location].url);
                    //将缓存的数据直接转发给客户端
                    ret = send(((ProxyParam*)lpParameter)->clientSocket,
Cache[last_cache_location].buffer, sizeof(Cache[last_cache_location].buffer), 0);
                    if (ret != SOCKET_ERROR)
                    {
                        printf("页面来自未修改过的缓存\n");
                    }
                }
                else if (strcmp(last_status, "200") == 0) {//如果页面已经已经修改了缓存中的内
容，状态码为200
                    printf("\n页面被修改过\n缓存的url为:%s\n",
Cache[last_cache_location].url);
                    memcpy(Cache[last_cache_location].buffer, cached_buffer,
strlen(cached_buffer));
                    memcpy(Cache[last_cache_location].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("页面来自修改过的缓存\n");
                    }
                }
            }
            //请求没有缓存的情况下
            else
            {
                //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: ParseCacheHead
        //FullName: ParseCacheHead
        //Access: public
        //Returns: void
        //Qualifier: 在cache命中的时候，解析cache中TCP报文中的HTTP头部
        //Parameter: char * buffer
        //Parameter: char * status
        //Parameter: HttpHeader *httpHeader
        //*************************
        void ParseCacheHead(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
	printf(p, "提取第一行  \n");
	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);
	}
}

//对禁止访问的网站表和钓鱼网站引导表进行处理
void replace(char buffer_c[], const string& oldstr, const string& newstr)
{
	string buffer = string(buffer_c);
	while (buffer.find(oldstr) != string::npos)    //如果buffer找到了oldstr循环
	{
		int m = buffer.find(oldstr);
		buffer = buffer.substr(0, m) + newstr + buffer.substr(m + oldstr.length());
	}
	memcpy(buffer_c, buffer.c_str(), buffer.length() + 1); //用新的网站地址替换原buffer_c
}

//************************************
// 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 = 0; //作为表示Cache是否命中的标志，命中为1，不命中为0
	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);//url

            for (int i = 0; i < 1024; i++) { //依次搜索缓存cache，确定当前访问的url是否已经存在cache中
                if (strcmp(Cache[i].url, httpHeader->url) == 0) { //当前url在已经存在cache中
                    flag = 1;
                    break;
                }
            }
            if (!flag && cached_quantities != 1023) {//当前url没有存在cache中，cache还存在空闲空间，往cache中存入url
                memcpy(Cache[cached_quantities].url, &p[4], strlen(p) - 13);
                last_cache_location = cached_quantities;
            }
            else if (!flag && cached_quantities == 1023) {//当前url没有存在cache中，但是cache已满,用该url覆盖第一个cache
                memcpy(Cache[0].url, &p[4], strlen(p) - 13);
                last_cache_location = 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++) { //依次搜索缓存cache，确定当前访问的url是否已经存在cache中
                if (strcmp(Cache[i].url, httpHeader->url) == 0) {
                    flag = 1;
                    break;
                }
            }
            if (!flag && cached_quantities != 1023) { //当前url没有存在cache中，cache还存在空闲空间，往cache中存入url
                memcpy(Cache[cached_quantities].url, &p[5], strlen(p) - 14);
                last_cache_location = cached_quantities;
            }
            else if (!flag && cached_quantities == 1023) { //当前url没有存在cache中，但是cache已满,用该url覆盖第一个cache
                memcpy(Cache[0].url, &p[4], strlen(p) - 13);
                last_cache_location = 0;
```

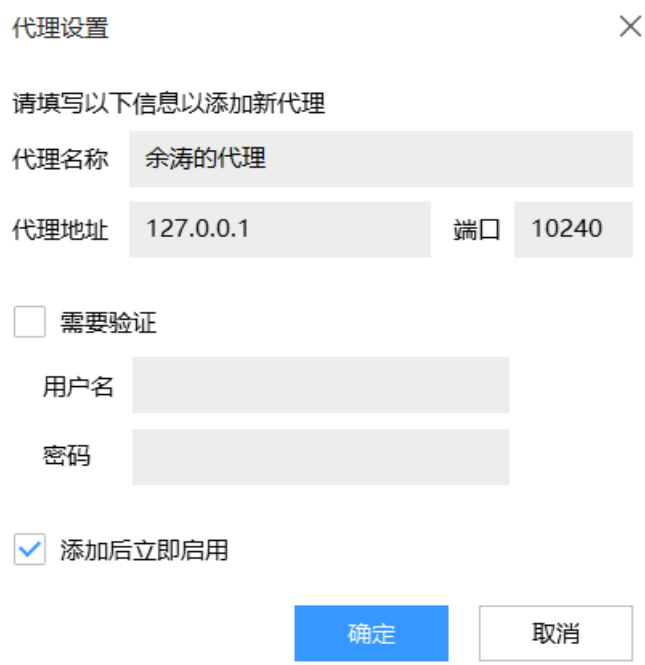
```
            }
        }
        //printf("%s\n", httpHeader->url);
        p = strtok_s(NULL, delim, &ptr);
        while (p) {
            switch (p[0]) {
            case 'H'://HOST
                memcpy(httpHeader->host, &p[6], strlen(p) - 6);
                if (!flag && cached_quantities != 1023) {
                    memcpy(Cache[last_cache_location].host, &p[6], strlen(p) - 6);
                    cached_quantities++;
                }
                else if (!flag && cached_quantities == 1023) {
                    memcpy(Cache[last_cache_location].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 (No_access_web_table.find(string(httpHeader->host)) != No_access_web_table.end())
        {
            printf("该网站 %s 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
        else if (Fishing_site_guide_table.find(string(httpHeader->host)) !=
Fishing_site_guide_table.end())
        {
            printf("引导至钓鱼网站 %s 成功\n", httpHeader->host);
            string target = Fishing_site_guide_table[string(httpHeader->host)];
```

```
                const char* target_c = target.c_str();
                replace(sendBuffer, string(httpHeader->host), target);    //用后一个host代替前一个host
                memcpy(httpHeader->host, target_c, 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;
        }
```

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.搜狗浏览器的代理服务器的设置：



2.运行程序：

3.基本HTTP代理服务器的实现：访问http://today.hit.edu.cn/

打印请求为：

```
C:\Users\Administrator\source\repos\1180300829_network_lab1\x64\Debug\1180300829_network_lab1.exe
650272D0OP-1022custom0X0.00000FFFFFD7FP-1022YD/Bdate0X0.000750C9DEFF0P-1022custom-0X1.CCCCCCCCCCCCCP+205mD/Bdate-0X1.CCCCCCCCCCCP+205custom0X1.4746820544547P+776dD/image-20201102132715-1.jpeg?itok=Nt28Thku HTTP/1.1
Host: today.hit.edu.cn
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36 SE 2.X MetaSr 1.0
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://today.hit.edu.cn/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: _ga=GA1.3.1538138047.1595739585; UM_distinctid=1757951161573-0430770b89562c-c7d6957-1fa400-17579511616193; _trs_uv=kgw6t8iq_4336_e18c; CNZZDATA1255991877=1433683774-1604055403-1604060823
Range: bytes=65131-98145
If-Range: "17f62-5b3199b458d33"

代理连接主机 today.hit.edu.cn成功
HTTP/1.1 206 Partial Content关闭套接字
请求内容为:
GET http://today.hit.edu.cn/sites/today1.prod1.dpweb1.hit.edu.cn/themes/hit_today/favicon.ico HTTP/1.1
Host: today.hit.edu.cn
Proxy-Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36 SE 2.X MetaSr 1.0
Accept-Encoding: gzip, deflate
Cookie: _ga=GA1.3.1538138047.1595739585; UM_distinctid=1757951161573-0430770b89562c-c7d6957-1fa400-17579511616193; _trs_uv=kgw6t8iq_4336_e18c; CNZZDATA1255991877=1433683774-1604055403-1604060823
```

4. Cache功能的实现：多次访问http://www.badong.net/

第一次访问：

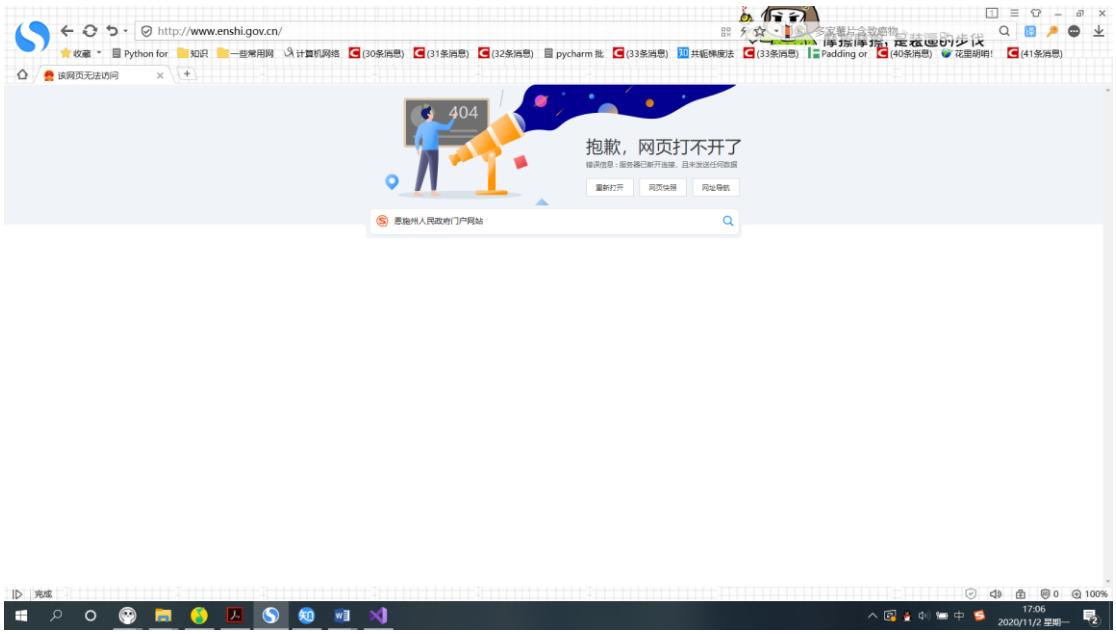

```
C:\Users\Administrator\source\repos\1180300829_network_lab1\x64\Debug\1180300829_network_lab1.exe
请求内容为:
GET http://www.badong.net/gbook/show.czfx?no-cache=0.9258705248678836&setJs_Act=showList&setJs_gbookCID=2&setJs_Style=2
HTTP/1.1
Host: www.badong.net
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.3
6 SE 2.X MetaSr 1.0
Accept: */*
Referer: http://www.badong.net/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: ASPSESSIONIDCQATBTDS=APNBOLCDNILHHNPCHAHFKNOA

代理连接主机 www.badong.net成功
关闭套接字
请求内容为:
GET http://www.badong.net/gbook/show.czfx?setJs_Act=showSearchTemp HTTP/1.1
Host: www.badong.net
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.3
6 SE 2.X MetaSr 1.0
Accept: */*
Referer: http://www.badong.net/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: ASPSESSIONIDCQATBTDS=APNBOLCDNILHHNPCHAHFKNOA
```

第二次访问：输出了304 Not Modified

```
代理连接主机 www.badong.net成功
HTTP/1.1 304 Not Modified
页面没有修改过
缓存的url为:http://www.badong.net/gbook/show.czfx?no-cache=0.9258705248678836&setJs_Act=showList&setJs_gbookCID=2&setJs_
Style=2
页面来自未修改过的缓存
关闭套接字
```

5. 网站过滤：禁止访问的网站为：http://www.enshi.gov.cn/

```
//禁止访问的网站表
set<string> No_access_web_table =
{
    "www.enshi.gov.cn",
    //"www.badong.net",
};
```



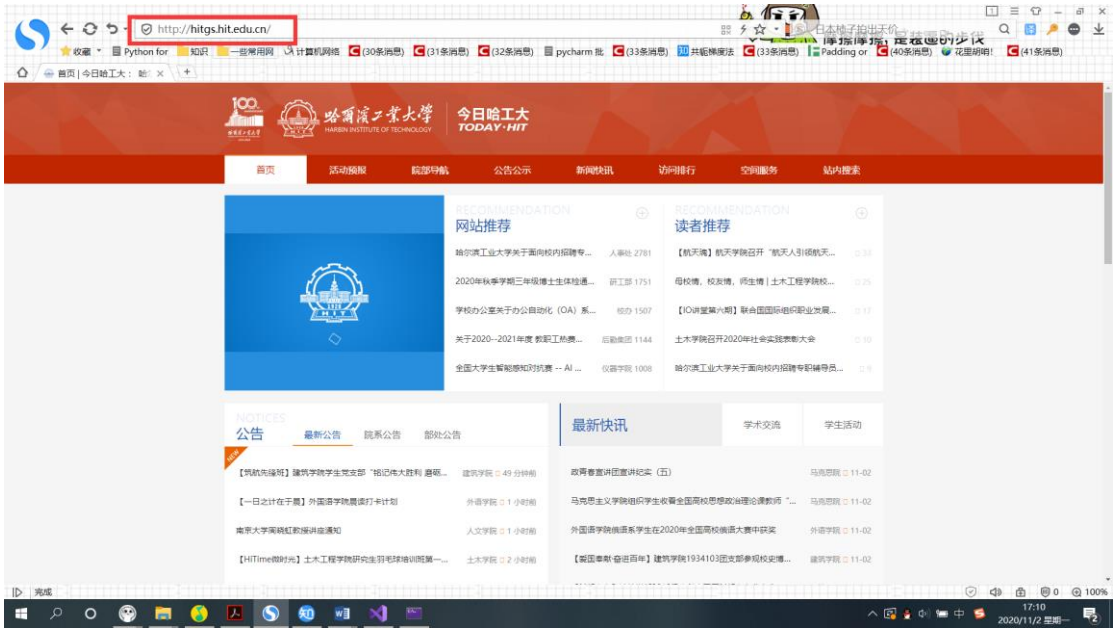

```
C:\Users\Administrator\source\repos\1180300829_network_lab1\x64\Debug\1180300829_network_lab1.exe

关闭套接字
请求内容为:
GET http://www.enshi.gov.cn/ HTTP/1.1
Host: www.enshi.gov.cn
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.3
6 SE 2.X MetaSr 1.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: UM_distinctid=17579140603cd-0a20f446cb592c-c7d6957-1fa400-175791406043be; _trs_uv=kgw4fgtj_4290_3rm; CNZZDATA125
5991877=2083669918-1604055403-null1604055403; Hm_lvt_bf6722103a40e3335c0f7daca20feefc=1604057366

该网站 www.enshi.gov.cn 禁止访问
请求内容为:
POST http://get.sogou.com/q HTTP/1.1
Host: get.sogou.com
Proxy-Connection: keep-alive
Content-Length: 618
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.3
6 SE 2.X MetaSr 1.0
Accept-Encoding: gzip, deflate
Cookie: SUID=89B69975B852A00A000000005EDF81E4; usid=UZAQYnKv4tBYNNOY; SUV=004B1DB27599B6895EDF881E5C4F1278; GOTO=Af12117
9; YYID=31C5B371B3FE96D28A9368E75477CCC7; pgv_pvi=7646889984; ssuid=8270661170; QIDIANID=yVs/Udv3CnrdoBX3xk9dqV4hT+3dUG7
6wv4cwpL5+E2oWnoNWzl7gnnbt3nEtqUG; sw_uuid=4748703719; ld=Gk111111112Wcp7B11111VMN5HU11111NBuLAy11117111114Z1115@@@@@@@@
@@; SNUID=748AA4A77C78CF6580D70BB97DFB2443; BAIDU_SSP_lcr=http://2345.ss8899888.com/; IPLOC=JP
```

6. 用户过滤：禁止访问的用户ip为：127.0.0.1（本机）

```
C:\Users\Administrator\source\repos\1180300829_network_lab1\x64\Debug\1180300829_network_lab1.exe

代理服务器正在启动
初始化...
代理服务器正在运行，监听端口 10240
用户 127.0.0.1没有权限，禁止访问该网站
用户 127.0.0.1没有权限，禁止访问该网站
用户 127.0.0.1没有权限，禁止访问该网站
```

7. 网站引导：将http://hitgs.hit.edu.cn/引导至http://today.hit.edu.cn/

```
//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
map<string, string> Fishing_site_guide_table =
{
    { "hitgs.hit.edu.cn", "today.hit.edu.cn" },
    { "", "" }
};
```



打印请求为：

```
C:\Users\Administrator\source\repos\1180300829_network_lab1\x64\Debug\1180300829_network_lab1.exe
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: _ga=GA1.3.1538138047.1595739585; UM_distinctid=1757951161573-0430770b89562c-c7d6957-1fa400-17579511616193; _trs_uv=kgw6t8iq_4336_e18c; CNZZDATA1255991877=1433683774-1604055403-1604060823
Range: bytes=65132-131429
If-Range: "20166-5b31a7dad818f"

代理连接主机 today.hit.edu.cn成功
HTTP/1.1 206 Partial Content关闭套接字
请求内容为：
GET http://hitgs.hit.edu.cn/sites/today1.prod1.dpweb1.hit.edu.cn/themes/hit_today/favicon.ico HTTP/1.1
Host: hitgs.hit.edu.cn
Proxy-Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36 SE 2.X MetaSr 1.0
Accept-Encoding: gzip, deflate
Cookie: _ga=GA1.3.1538138047.1595739585; UM_distinctid=1757951161573-0430770b89562c-c7d6957-1fa400-17579511616193; _trs_uv=kgw6t8iq_4336_e18c

引导至钓鱼网站 hitgs.hit.edu.cn 成功
代理连接主机 today.hit.edu.cn成功
HTTP/1.1 200 OK
页面被修改过
缓存的url为:http://today.hit.edu.cn/sites/today1.prod1.dpweb1.hit.edu.cn/files/styles/355x220/public/inline-images/%5Bdate%3Acustom%3AY%5D/%5Bdate%3Acustom%3Am%5D/%5Bdate%3Acustom%3Ad%5D/image_2544.png?h=e588d01a&itok=LFvA0n9x
```

问题讨论：

对实验过程中的思考问题进行讨论或回答。

1.使用微软自带的Edge或者谷歌浏览器运行程序后会不断重复关闭套接字的过程，所以改为尝试搜狗浏览器进行实验。

2.goto语句后面不能定义新的变量，会在代码块中报错。

3.127.0.0.1的ip地址被复制为localhost，即本机地址。当IP层接受到目的地址为127.0.0.1的数据包后，不调用网卡驱动进行二次封装，而是立即转发到本机IP层进行处理，不涉及到底层操作。

心得体会：
结合实验过程和结果给出实验的体会和收获。 本次实验，让我对socket编程有了初步的了解，掌握了 HTTP 代理服务器的基本原理经过此次实验，清楚客户端和服务器之间具体的Socket通信过程。对 HTTP 请求和响应原理有了更深的认识；对网站钓鱼、网站屏蔽以及 Cache等的原理等有了深刻的理解。