哈尔滨工业大学

<<计算机网络>>

实验报告

(2017 年度春季学期)

姓名:	赵浩宁
学号:	1140310226
学院:	计算机科学与技术学院
教师:	聂兰顺

实验一 HTTP 代理服务器的设计与实现

一、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。

二、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（钓鱼网站）

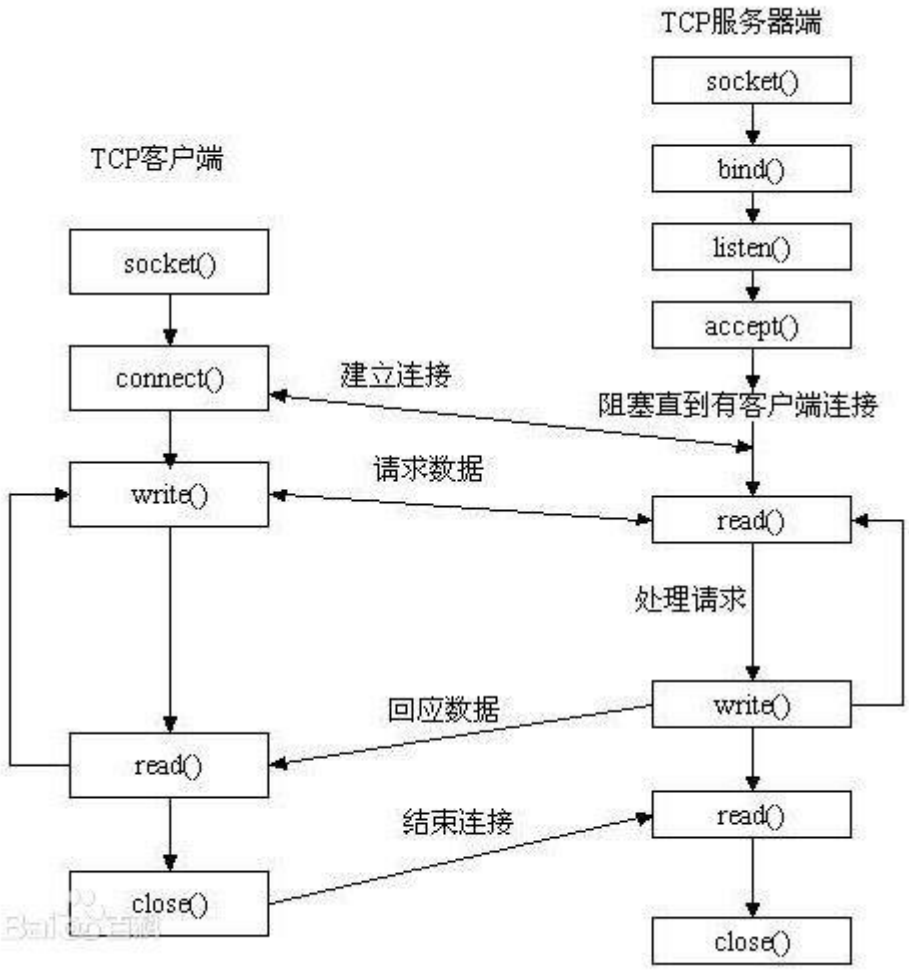
三、实验过程及结果

1、实验原理

(1) Socket 编程的客户端和服务器端主要步骤

在 TCP/IP 网络应用中，通信的两个进程之间相互作用的主要模式是客户/服务器（C/S 或 B/S）模式，即客户向服务器发出请求，服务器接受到请求后，提供相应的服务。

两者的工作步骤可以通过下面流程图直观地看到：



服务器端：

其过程是首先服务器方要先启动，并根据请求提供相应服务：

（1）打开一通信通道并告知本地主机，它愿意在某一公认地址上的某端口接收客户请求；对应的操作是申请一个 socket，这时的 socket 称为“欢迎套接字”，然后绑定（bind）本地地址信息和欢迎套接字，然后开放监听（listen）。

（2）等待客户请求到达该端口；

（3）接收到客户端的服务请求时，处理该请求并发送应答信号。在 TCP 实现过程中进行了三次握手操作，但是实际编写过程中通过 accept 函数即可实现上述操作，并建立连接，注意这个时候才真正建立起了与客户机传输数据的套接字。接收到并发服务请求，要激活一新进程来处理这个客户请求。新进程处理此客户请求，并不需要对其它请求作出应答。服务完成后，关闭此新进程与客户的通信链路，并终止。

（4）返回第（2）步，等待另一客户请求。

（5）关闭服务器，对应的也就是关闭服务器的欢迎套接字。

客户端：

（1）打开一通信通道，即建立起要与服务器传输数据的套接字 socket，通过 connect 连接到服务器所在主机的特定端口；

（2）向服务器发服务请求报文，等待并接收应答；继续提出请求.....

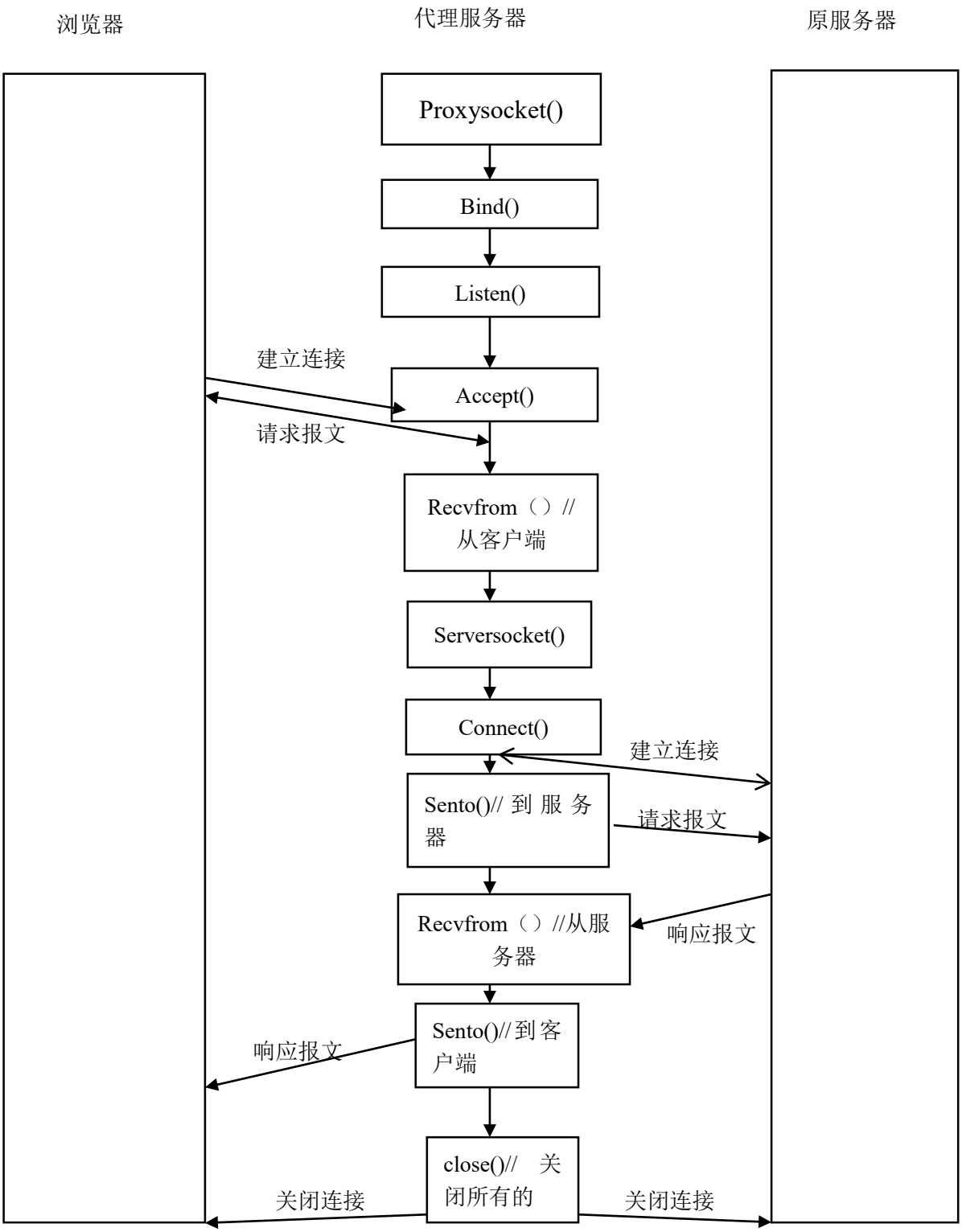
（3）请求结束后关闭通信通道并终止。

从上面所描述过程可知：

（1）客户与服务器进程的作用是非对称的，因此代码不同。

（2)服务器进程一般是先启动的。只要系统运行，该服务进程一直存在，直到正常或强迫终止。

流程图：



(2) HTTP 代理服务器的基本原理与流程图

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 1-1 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。

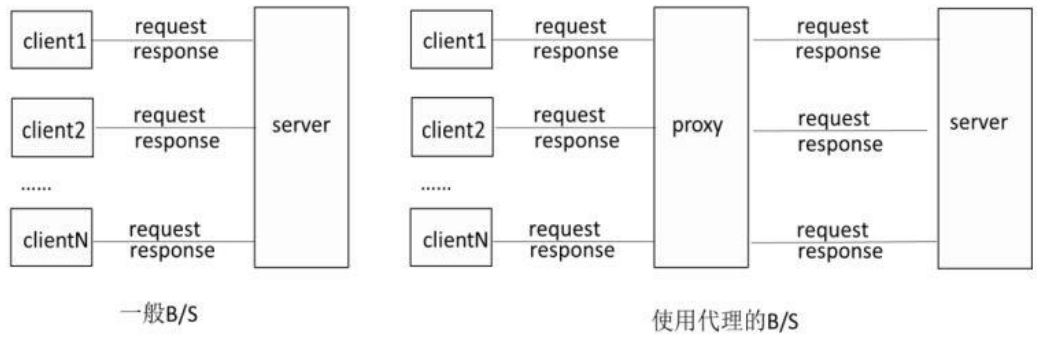


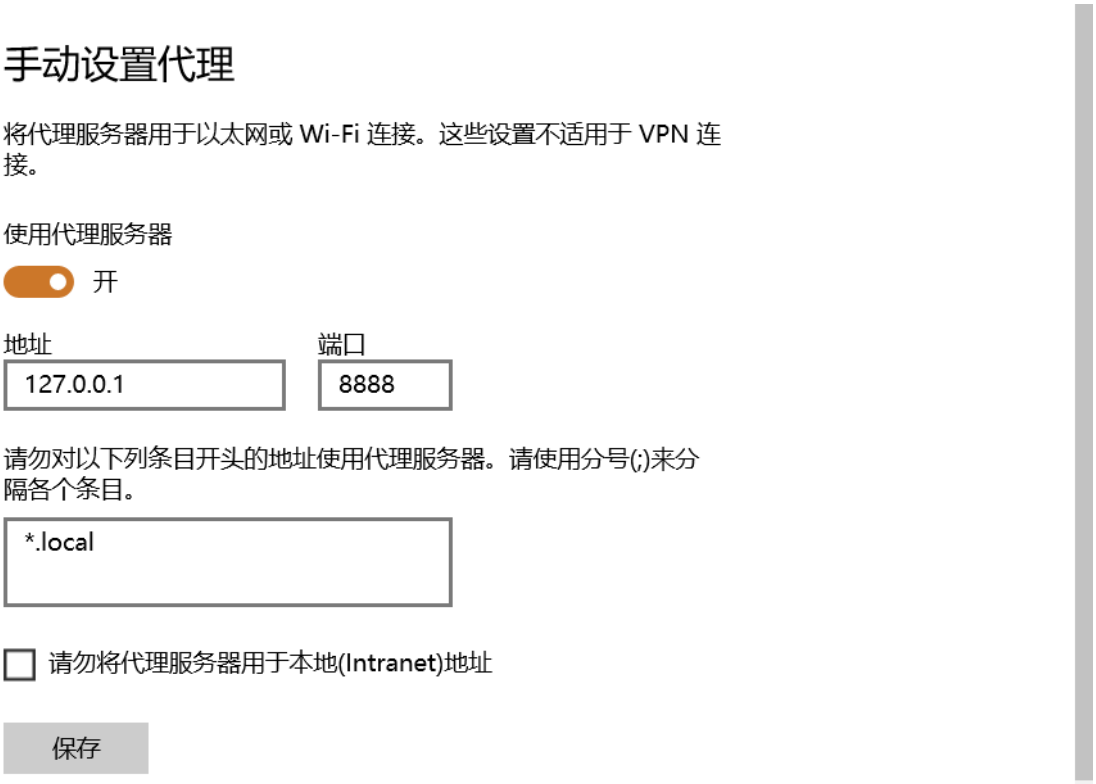
图 1-1

代理服务器可以认为是 TCP/IP 网络应用的客户端和服务器端的结合。一方面，它是浏览器客户端的服务器端，另一方面，它也是目标服务器的客户端。浏览器将请求报文发送给代理服务器，代理服务器经过一些处理或者不经过处理，将请求报文转发给目标服务器；目标服务器相应请求报文发出响应报文，代理服务器接受到响应报文之后直接将响应报文转发给浏览器客户端。

代理服务器在指定端口（例如 8888）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，首先查看浏览器来源的 ip 地址，如果属于被限制的用户，则认为没有接受到访问请求。否则，查看其请求的 host 主机，如果属于不允许访问的主机，则默认不向目标服务器发送请求；如果属于被引导的网站，则对该网站的请求报文中的 host 主机地址和 url 进行更改，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。

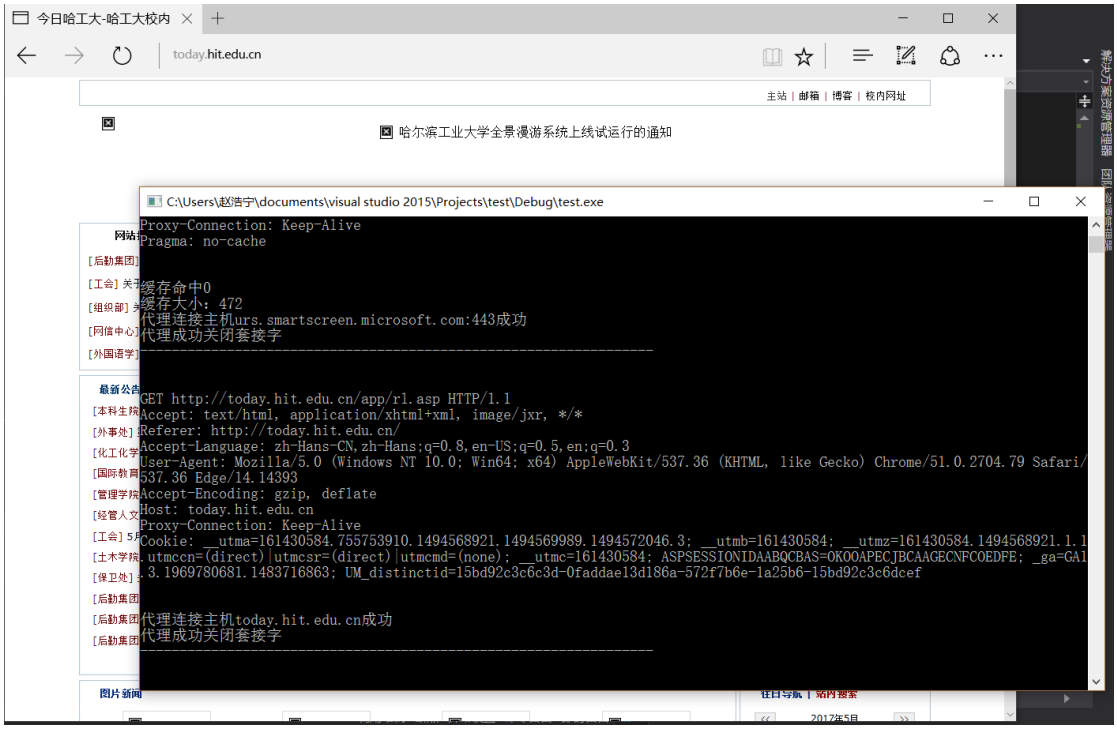
(3) HTTP 代理服务器实验验证过程以及实验结果

1．设置 IE 浏览器的代理服务器



2．运行程序

3．在 IE 浏览器输入 www.today.hit.edu.cn，在程序运行窗口发现了请求报文，并在浏览器端接收到了网页的数据，说明代理服务器基本功能实现。



4.（拓展功能 1-支持 Cache 功能）

原理：

1.先判断缓存是否命中

2.若没有命中则直接请求对象，获取对象后，查看 cache-control 字段

3.若可以缓存，则保存到本地（完整的报文），并发送回客户端

4.若不可以缓存，则直接发送回客户端

5.若缓存命中（是否过期：DATE+max-age 与当前时间的对比）

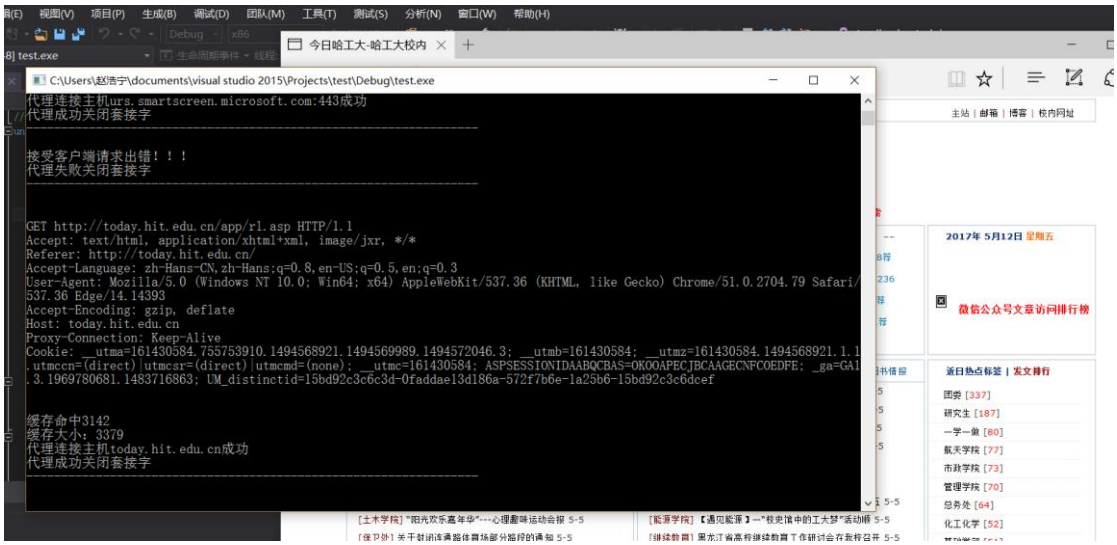
6.若没有过期，直接从缓存中读取，并发回客户端

7.若过期，向服务器发送 IF-MODIFIED-SINCE：LAST MODIFIED 中的时间

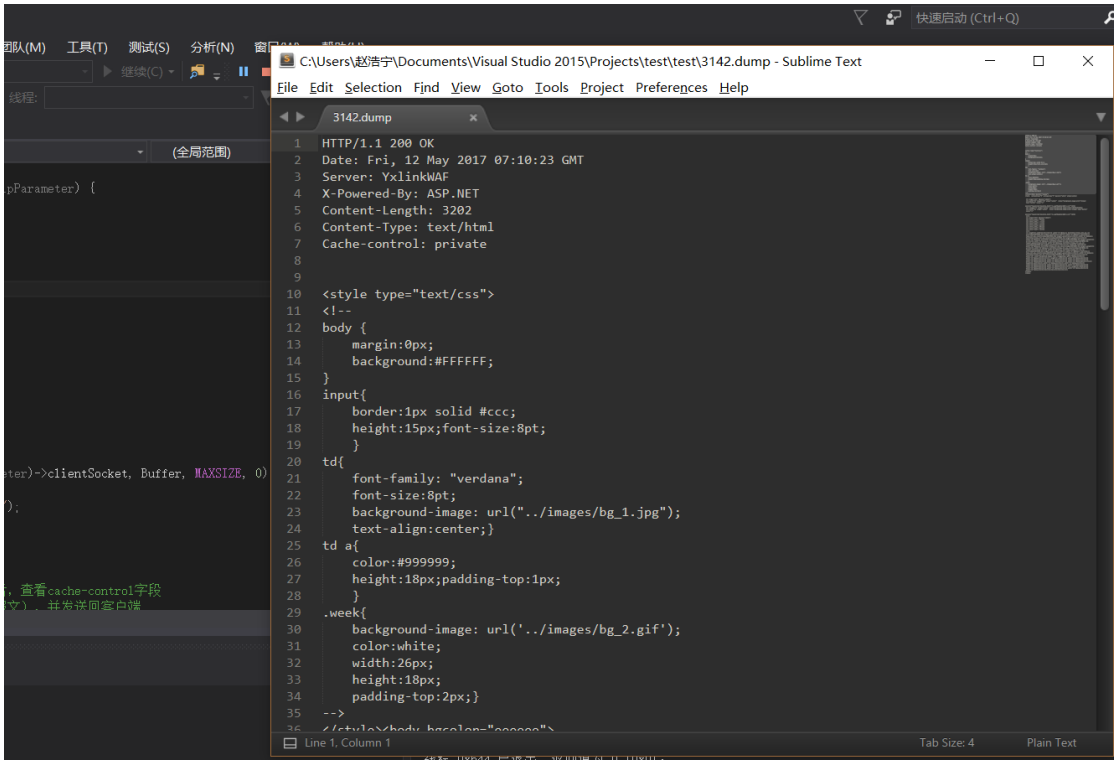
8.若返回 200，则更新缓存，并返回客户端

9.若返回 304，返回客户端

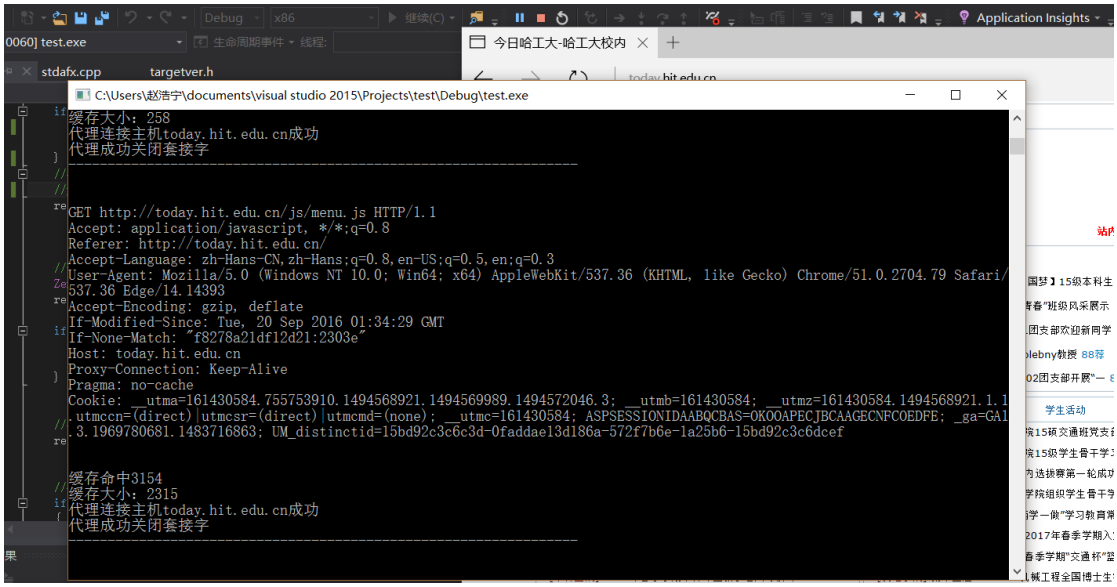
再次访问 www.today.hit.edu.cn，此时从缓存文件中进行读取，缓存命中，代理服务器检查缓存是否过期，然后将数据报文发送给客户端。



可以查看文件 3142 中的内容，可以看到是 www.today.hit.edu.cn 在本地的一个内容备份

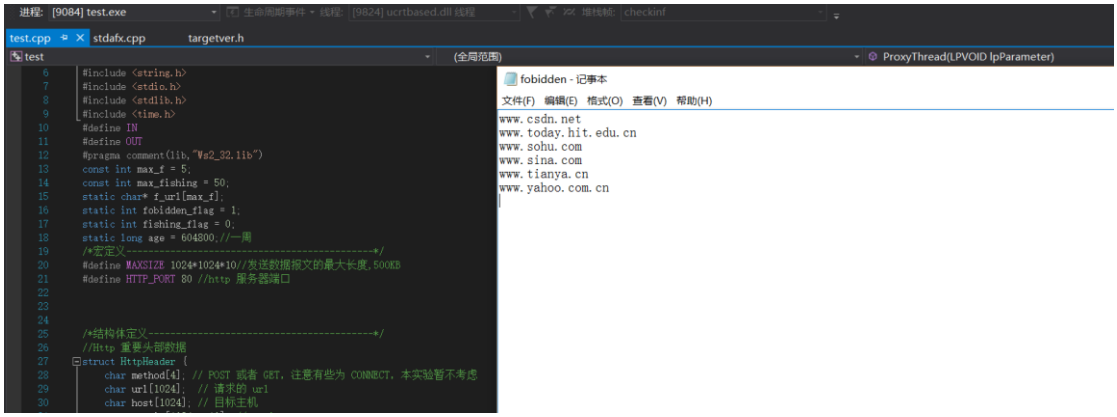


过一段时间后刷新网页，客户端会重新发送 http 请求，代理服务器接受到请求后发现缓存命中，此时要向服务器发送请求，查看 IF-Modified-Since 字段，与服务器进行比较，如果缓存过期了，则要更新缓存，并将数据发送给客户端

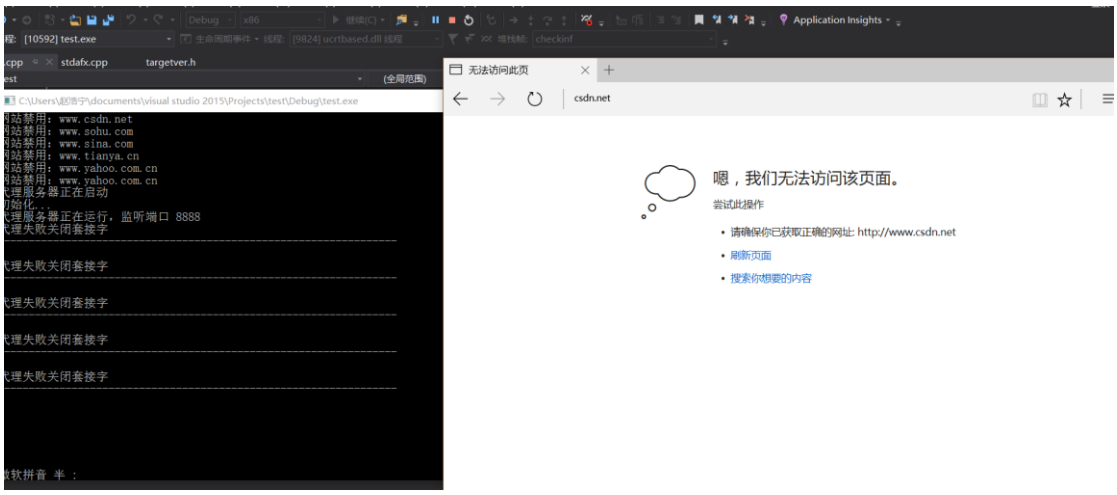


5.（拓展功能 2-a-网站过滤）

在 **forbidden.txt** 中定义要禁止访问的网站，并在宏定义中将 **forbidden** 置为 **1**，则对于文件中的网站都会禁止用户访问



对于禁止访问的网站，直接返回 error

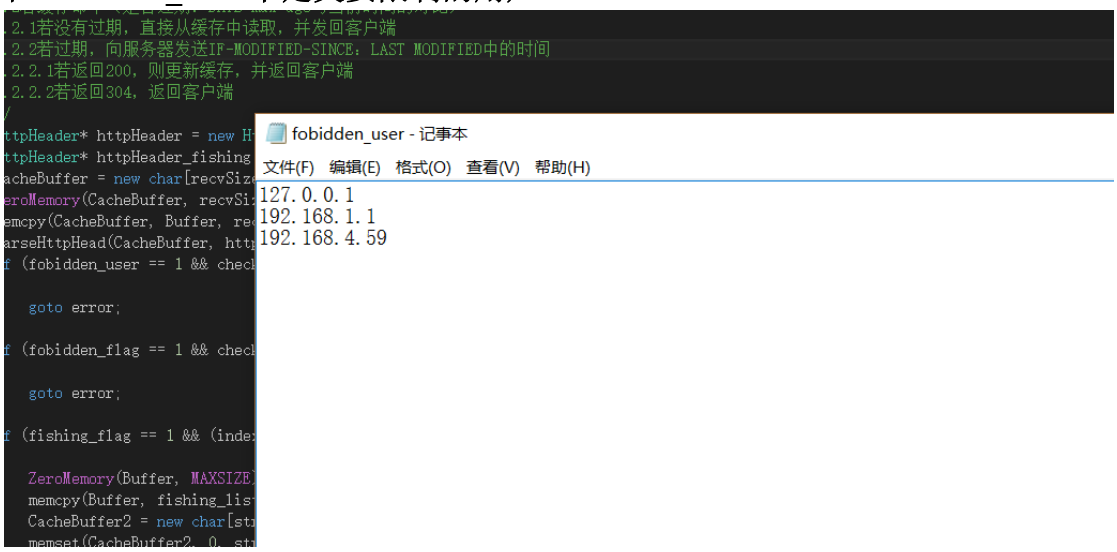


我们发现网站受限，无法通过代理访问

6.（拓展功能 2-b-用户过滤）

添加 fobidden_user=1 的宏定义

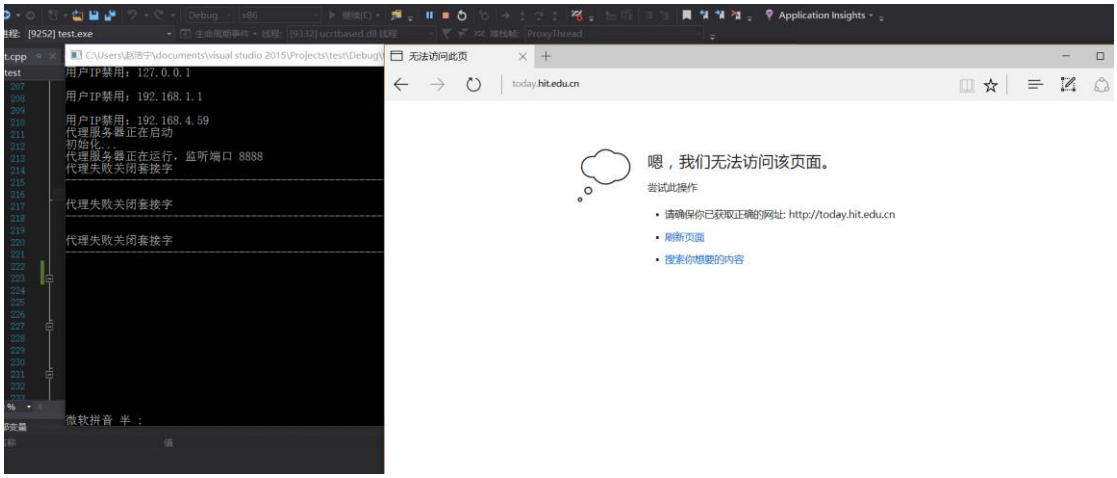
在 fobidden_user 中定义要限制的用户 IP



读取要限制的 IP

```
void read_user(char path[])
{
    for (int i = 0; i < max_f; i++)
    {
        f_user[i] = NULL;
    }
    int i = 0;
    FILE *fp = fopen(path, "r");
    char Strur1[1024] = { 0 };
    if (fp == NULL)
    {
        printf("打开文件失败\n");
    }
    while (!feof(fp))
    {
        fgets(Strur1, 1024, fp);
        f_user[i] = new char[strlen(Strur1) + 1];
        ZeroMemory(f_user[i], strlen(Strur1) + 1);
        memcpy(f_user[i], Strur1, strlen(Strur1));
        printf("用户IP禁用：%s\n", f_user[i]);
        i++;
    }
```

对于限制的 IP 代理服务器不响应请求，所以无法访问网站



7.（拓展功能 2-c-网站引导）

在 fishing.txt 中设置要钓鱼的客户请求 url

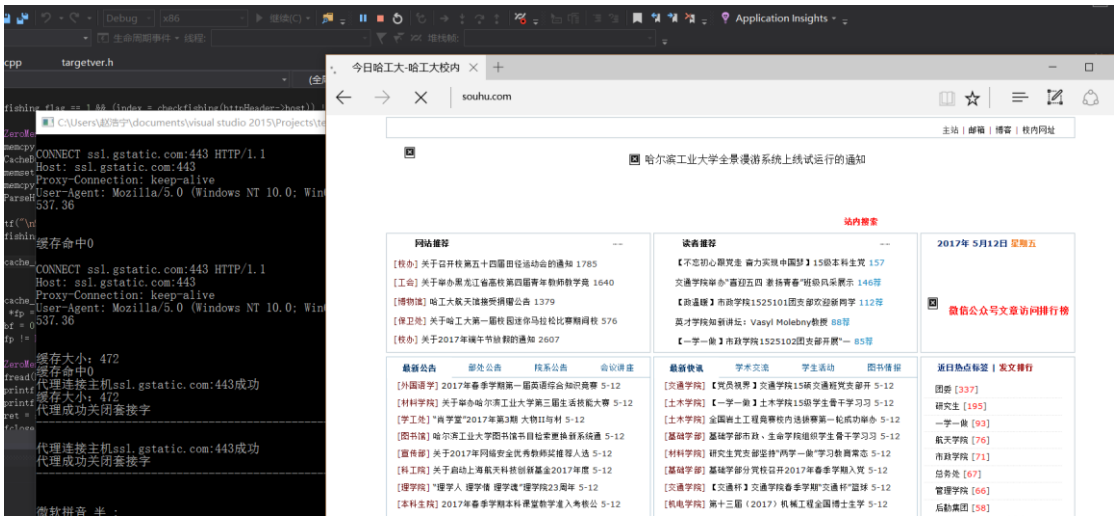
```
(fishing_flag == 1 && (index = checkfishing(httpHeader->host)) != -1)

 ZeroMemory(Buffer, MAXSIZE);
 memcpy(Buffer, fishing_list[index].fishing_head, strlen(fishing_list[index].fishing_head));
 CacheBuffer2 = new char[strlen(Buffer)];
 memset(CacheBuffer2, 0, strlen(Buffer));
 memcpy(CacheBuffer2, Buffer, strlen(Buffer));
 ParseHttpHead(CacheBuffer2, httpHeader_fishing);

intf("\n%s\n", Buffer);
 (fishing_flag == 1 && index != -1)

  cache_dir = "http:\\www.today.hit.edu.cn\\";

se
  cache_dir = makechar(httpHeader->url);
LE *fp = fopen(cache_dir, "rb");//cache中该缓存是否存在
t bf = 0;
 (fp != NULL && checkoutofdate(fp) == 0)//缓存命中且没有过期

 ZeroMemory(Buffer, MAXSIZE);
 fread(Buffer, sizeof(char), MAXSIZE, fp);
 printf("缓存命中%s\n", cache_dir);
```

fishing - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
www.pku.edu.cn
www.sina.com
www.hit.edu.cn
www.sohu.com
!
```

当输入 www.sohu.com 以后，网站被重定向到了 www.today.hit.edu.cn



成功引导

(4)实现 HTTP 代理服务器的关键技术及解决方案

简单代理服务器通过以下几个函数实现：

1.

```
//**************************************
// Method:        InitSocket
// FullName:    InitSocket
// Access:        public
// Returns:      BOOL
// Qualifier:  初始化套接字
//**************************************
BOOL InitSocket()
```

该函数首先加载套接字库，这一步是必须的；然后分别使用 C 的库函数里的 socket(AF_INET, SOCK_STREAM, 0);

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

实现了服务器流程中的 socket 和 bind 和 listen；

2.

```
//**************************************
// 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)
```

该函数对请求报文的头部文件进行解析，得到请求报文中的 method,url,host 和 cookie 存到一个对应的结构体中，该结构体用于

ConnectToServer 函数与原服务器建立链接。

3.

```
//************************************
// 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)
```

实现与服务器之间的连接，

4.

```
//************************************
// 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)
```

该函数是核心函数，其实现了从浏览器接收请求报文，向服务器发送请求报文，从服务器接收响应报文，向浏览器发送响应报文。

基本的代理服务器中没有缓冲，处理方式中仅对请求报文头部进行解析，通过 ParseHttpHead 函数，然后将得到的头部文件作为 ConnectToServer 函数与原服务器建立链接。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。

5. 代理服务器设置 cahce 实现方式

```
int bf = 0;
if (fp != NULL && checkoutofdate(fp) == 0)//缓存命中且没有过期
{
    ZeroMemory(Buffer, MAXSIZE);
    fread(Buffer, sizeof(char), MAXSIZE, fp);
    printf("缓存命中%s\n", cache_dir);
    printf("缓存大小：%d\n", strlen(Buffer));
    ret = send(((ProxyParam *)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);//将缓存的报文回传
    fclose(fp);
    goto success;//结束该线程
}
else if (checkoutofdate(fp) == 1)//过期
{
    addifsincemodified(Buffer, httpHeader);
```

```
603     }
604     void addifsincemodified(char* buffer, HttpHeader* httpHeader) {
605         char add[1024] = { 0 };
606         memcpy(add, "if modified since", 20);
607         strcat(add, httpHeader->date);
608         strcat(buffer, add);
609     }
610
```

在缓冲区中找存储的请求报文的头部，buffer是该请求报文在服务器端返回的响应报文，date指该响应报文最后一次更新的时间，即该响应报文中的last-modifited。

然后在ProxyThread函数中，当收到请求报文时，在对报文头部处理之后，首先在cache中寻找该请求，如果找到了，在请求报文之中-第三行加入if-modified-since: date，发送给服务器，接收到服务器返回的响应报文，对响应报文进行处理，看其头部是否为304 not modified，如果是，直接将cache中的响应报文返回给浏览器，如果不是，首先将该响应报文存入cache中，即对cache进行更新—仍存储在之前的那个位置上，然后将响应报文返回给浏览器。如果在cache中没有找到该请求，将处理后的请求报文头部存入Cache，得到响应报文之后，对响应报文进行解析，得到date，然后将响应报文和date存入cache。

```
int checkoutofdate(FILE* fp) {
    char date[40] = { 0 };
    long second = 0;
    long nowt = 0;
    struct tm *nowtime = (struct tm*)malloc(sizeof(struct tm));
    struct tm *time1 = (struct tm*)malloc(sizeof(struct tm));
    if (fp == NULL || 1)
    {
        return 0;
    }
    else
    {
        char strline[200] = { 0 };
        while (fp != NULL)
        {
            fgets(strline, 1024, fp);
            if (strline[0] == 'D'&&strline[1] == 'a'&&strline[2] == 't'&&strline[3] == 'e')
            {
                memcpy(date, &strline[6], 25);
                timeconvert(date, time1);
                second = mktime(time1);
                break;
            }
        }
        if (nowt - second < age)
```

检查缓存是否过期

6. 网站过滤

```
476 }
477 void read_void(char path[])
478 {
479     for (int i = 0; i < max_f; i++)
480     {
481         f_url[i] = NULL;
482     }
483     int i = 0;
484     FILE *fp = fopen(path, "r");
485     char Strur1[1024] = { 0 };
486     if (fp == NULL)
487     {
488         printf("打开文件失败\n");
489     }
490     while (!feof(fp))
491     {
492         fgets(Strur1, 1024, fp);
493         f_url[i] = new char[strlen(Strur1) + 1];
494         ZeroMemory(f_url[i], strlen(Strur1) + 1);
495         memcpy(f_url[i], Strur1, strlen(Strur1));
496         printf("网站禁用：%s", f_url[i]);
497         i++;
498     }
499     fclose(fp);
500 }
```

```
    }
    if (fobidden_flag == 1 && checkinf(httpHeader->host) == 1)
    {
        goto error;
    }
    if (fishing_flag == 1 && (index = checkfishing(httpHeader->host)) != -1)
```

读取禁用的网络url，添加到数组里，当访问的时候直接返回异常

7. 用户过滤

在主函数中，当建立起浏览器和代理服务器的链接时，得到浏览器的地址信息，也就得到浏览器端的ip地址，与被禁用户ip比较，如果相同，认为链接没有建立，代理服务器等待下一次访问请求。

```
void read_user(char path[])
{
    for (int i = 0; i < max_f; i++)
    {
        f_user[i] = NULL;
    }
    int i = 0;
    FILE *fp = fopen(path, "r");
    char Strur1[1024] = { 0 };
    if (fp == NULL)
    {
        printf("打开文件失败\n");
    }
    while (!feof(fp))
    {
        fgets(Strur1, 1024, fp);
        f_user[i] = new char[strlen(Strur1) + 1];
        ZeroMemory(f_user[i], strlen(Strur1) + 1);
        memcpy(f_user[i], Strur1, strlen(Strur1));
        printf("用户IP禁用：%s\n", f_user[i]);
        i++;
    }
    fclose(fp);
}
```

8. 网站引导

在ProxyThread函数中解析出请求报文头部之后，将请求报文头部中的url与被引导的网站比较，如果相同，将请求报文中的url改为引导向的网站的url，host也变为引导向的网站的host即可。

```
if (fishing_flag == 1 && (index = checkfishing(httpHeader->host)) != -1)
{
    ZeroMemory(Buffer, MAXSIZE);
    memcpy(Buffer, fishing_list[index].fishing_head, strlen(fishing_list[index].fishing_head));
    CacheBuffer2 = new char[strlen(Buffer)];
    memset(CacheBuffer2, 0, strlen(Buffer));
    memcpy(CacheBuffer2, Buffer, strlen(Buffer));
    ParseHttpHead(CacheBuffer2, httpHeader_fishing);
}
printf("\n%s\n", Buffer);
if (fishing_flag == 1 && index != -1)
{
    cache_dir = "http:\\www.today.hit.edu.cn\\";
}
else
    cache_dir = makechar(httpHeader->url);
FILE *fp = fopen(cache_dir, "rb");//cache中该缓存是否存在
```

四、实验心得

经过此次实验，在实践过程中学到了 TCP 协议在传输数据的流程和方式；熟悉了 Socket 网络编程的过程与技术；同时也更清晰地掌握了 HTTP 代理服务器的基本工作原理；掌握了 HTTP 代理服务器设计与编程实现的基本技能。并且了解了浏览器在进行搜索网页过程中，网络所起的作用以及具体的工作原理，数据的传输方式等等，受益匪浅。

在设计 Cache 和网站过滤、网站引导和用户过滤的过程中，也了解到了很多 C 语言 scoket 编程中函数的一些用法，一些技巧，是学习和掌握网络中应用层数据传输方式的很有效的方式。

五、源代码

```
/*头文件---------------------------------------------*/
#include "stdafx.h"
#include <Windows.h>
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define IN
#define OUT
#pragma comment(lib,"Ws2_32.lib")
const int max_f = 100;
const int max_fishing = 50;
static char* f_url[max_f];
static char* f_user[max_f];
static int fobidden_flag = 0;
static int fobidden_user = 0;
static int fishing_flag = 0;
static long age = 604800;//一周
/*宏定义---------------------------------------------*/
#define MAXSIZE 1024*1024*10//发送数据报文的最大长度,500KB
#define HTTP_PORT 80 //http  服务器端口
```

```
/*结构体定义-----------------------------------------*/
//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
	char date[1024];
	HttpHeader() {
		ZeroMemory(this, sizeof(HttpHeader));
	}
};
struct ProxyParam {
	SOCKET clientSocket;
	SOCKET serverSocket;
};
struct fishing {
	char init_host[50];
	char fishing_host[50];
	char fishing_head[500];
};


/*函数声明-----------------------------------------------*/
BOOL InitSocket();
void ParseHttpHead(char *buffer, HttpHeader * httpHeader);
BOOL ConnectToServer(SOCKET *serverSocket, char *host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
int checkinf(char *host);
int checkuser(char* user);
void read_void(char path[]);
void read_user(char path[]);
void read_fishing(char path[]);
int checkfishing(char* host);
char* makechar(char* name);
int checknum(char* buffer);
int checkoutofdate(FILE* fp);
void addifsincemodified(char* buffer, HttpHeader* httpHeader);
time_t timeconvert(IN char *buf, OUT struct tm *p);
int monthcmp(IN char *p);
int weekcmp(IN char *p);
```

```
/*全局变量声明-----------------------------------------*/
//代理相关参数
SOCKET ProxyServer;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 8888;
struct fishing fishing_list[max_fishing];
char *me = "127.0.0.1";
//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};

//CRITICAL_SECTION g_cs;
int _tmain(int argc, _TCHAR* argv[])
{
	//InitializeCriticalSection(&g_cs);	//
	if (fobidden_user == 1)
	{
		read_user("fobidden_user.txt");
	}
	if (fobidden_flag == 1)
	{
		read_void("fobidden.txt");
	}
	if (fishing_flag == 1)
	{
		read_fishing("fishing.txt");
	}
	printf("代理服务器正在启动\n");
	printf("初始化...\n");
	if (!InitSocket()) {
		printf("socket  初始化失败\n");
		return -1;
	}
	printf("代理服务器正在运行，监听端口  %d\n", ProxyPort);
	SOCKET acceptSocket = INVALID_SOCKET;
	ProxyParam *lpProxyParam;
	HANDLE hThread;
	DWORD dwThreadID;
	//代理服务器不断监听
	while (true) {
		acceptSocket = accept(ProxyServer, NULL, 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);
        }
        //DeleteCriticalSection(&g_cs);
        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 TCP 套接字
```

```
    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, 3000) == 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) {
    //EnterCriticalSection(&g_cs);
    char *Buffer;
    char *CacheBuffer;
    char *CacheBuffer2;
    Buffer = new char[MAXSIZE];
    ZeroMemory(Buffer, MAXSIZE);
    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    char ch;
    int index;
    char *cache_dir;
    cache_dir = new char[100];
    memset(cache_dir, 0, 100);
```

```
    //从客户端接收 http 请求
    recvSize = recv(((ProxyParam *)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
    if (recvSize <= 0) {
        printf("接受客户端请求出错！！！\n");
        goto error;
    }
    /*
    1.先判断缓存是否命中
    1.1 若没有命中则直接请求对象，获取对象后，查看 cache-control 字段
    1.1.1 若可以缓存，则保存到本地（完整的报文），并发送回客户端
    1.1.2 若不可以缓存，则直接发送回客户端
    1.2 若缓存命中（是否过期：DATE+max-age 与当前时间的对比）
    1.2.1 若没有过期，直接从缓存中读取，并发回客户端
    1.2.2 若过期，向服务器发送 IF-MODIFIED-SINCE：LAST MODIFIED 中的时间
    1.2.2.1 若返回 200，则更新缓存，并返回客户端
    1.2.2.2 若返回 304，返回客户端
    */
    HttpHeader* httpHeader = new HttpHeader();
    HttpHeader* httpHeader_fishing = new HttpHeader();
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);
    ParseHttpHead(CacheBuffer, httpHeader);
    if (fobidden_user == 1 && checkuser(me) == 1)
    {
        goto error;
    }
    if (fobidden_flag == 1 && checkinf(httpHeader->host) == 1)
    {
        goto error;
    }
    if (fishing_flag == 1 && (index = checkfishing(httpHeader->host)) != -1)
    {
        ZeroMemory(Buffer, MAXSIZE);
        memcpy(Buffer,                                    fishing_list[index].fishing_head,
strlen(fishing_list[index].fishing_head));
        CacheBuffer2 = new char[strlen(Buffer)];
        memset(CacheBuffer2, 0, strlen(Buffer));
        memcpy(CacheBuffer2, Buffer, strlen(Buffer));
        ParseHttpHead(CacheBuffer2, httpHeader_fishing);
    }
    printf("\n%s\n", Buffer);
    if (fishing_flag == 1 && index != -1)
    {
```

```
            cache_dir = "http:\\www.today.hit.edu.cn\\";
        }
        else
            cache_dir = makechar(httpHeader->url);
        FILE *fp = fopen(cache_dir, "rb");//cache 中该缓存是否存在
        int bf = 0;
        if (fp != NULL && checkoutofdate(fp) == 0)//缓存命中且没有过期
        {
            ZeroMemory(Buffer, MAXSIZE);
            fread(Buffer, sizeof(char), MAXSIZE, fp);
            printf("缓存命中%s\n", cache_dir);
            printf("缓存大小：%d\n", strlen(Buffer));
            ret = send(((ProxyParam *)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);//将缓存
的报文回传
            fclose(fp);
            goto success;//结束该线程
        }
        else if (checkoutofdate(fp) == 1)//过期
        {
            addifsincemodified(Buffer, httpHeader);
        }
        //如果没有命中缓存

        if (!ConnectToServer(&((ProxyParam *)lpParameter)->serverSocket, httpHeader->host)) {
            printf("连接服务器出错\n");
            goto error;
        }
        //将客户端发送的 HTTP 数据报文直接转发给目标服务器
        //printf("\n%s\n", Buffer);
        ret = send(((ProxyParam *)lpParameter)->serverSocket, Buffer, strlen(Buffer), 0);



        //等待目标服务器返回数据
        ZeroMemory(Buffer, MAXSIZE);
        recvSize = recv(((ProxyParam *)lpParameter)->serverSocket, Buffer, MAXSIZE, 0);

        if (recvSize <= 0) {
            printf("接受服务器返回错误!!! \n");
            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, strlen(Buffer), 0);


    //将数据存入本地作为缓存
    if (checknum(Buffer) == 200)
    {
        FILE *fm = fopen(cache_dir, "wb");//缓存文件，以 URL 作为文件名
        if (fm != NULL)
        {
            fwrite(Buffer, sizeof(char), strlen(Buffer), fm);
            fclose(fm);
        }
    }
    goto success;
error:
    printf("代理失败关闭套接字\n");
    printf("--------------------------------------------------------------\n\n");
    closesocket(((ProxyParam*)lpParameter)->clientSocket);
    closesocket(((ProxyParam*)lpParameter)->serverSocket);
    delete lpParameter;
    delete Buffer;
    _endthreadex(0);
    return 0;
success:
    printf("代理连接主机%s 成功\n", httpHeader->host);
    printf("代理成功关闭套接字\n");
    printf("--------------------------------------------------------------\n\n");
    closesocket(((ProxyParam*)lpParameter)->clientSocket);
    closesocket(((ProxyParam*)lpParameter)->serverSocket);
    delete lpParameter;
    delete Buffer;
    _endthreadex(0);
    return 0;
}

//*************************************
// Method: ParseHttpHead
// FullName: ParseHttpHead
// Access: public
// Returns: void
// Qualifier: 解析 TCP 报文中的 HTTP 头部
// Parameter: char * buffer
// Parameter: HttpHeader * httpHeader
//*************************************
```

```
void ParseHttpHead(char *buffer, HttpHeader * httpHeader) {
	char *p;
	char *ptr;
	const char * delim = "\r\n";
	p = strtok_s(buffer, delim, &ptr);//提取第一行
	//printf("%s\n", p);
	if (p[0] == 'G') {//GET 方式
		memcpy(httpHeader->method, "GET", 3);
		memcpy(httpHeader->url, &p[4], strlen(p) - 13);
	}
	else if (p[0] == 'P') {//POST 方式
		memcpy(httpHeader->method, "POST", 4);
		memcpy(httpHeader->url, &p[5], strlen(p) - 14);
	}
	//printf("%s\n", httpHeader->url);
	p = strtok_s(NULL, delim, &ptr);
	while (p) {
		switch (p[0]) {
		case 'H'://Host
			memcpy(httpHeader->host, &p[6], strlen(p) - 6);
			break;
		case 'C'://Cookie
			if (strlen(p) > 8) {
				char header[8];
				ZeroMemory(header, sizeof(header));
				memcpy(header, p, 6);
				if (!strcmp(header, "Cookie")) {
					memcpy(httpHeader->cookie, &p[8], strlen(p) - 8);
				}
			}
			break;
		default:
			break;
		}
		p = strtok_s(NULL, delim, &ptr);
	}
}
//************************************
// Method: ConnectToServer
// FullName: ConnectToServer
// Access: public
// Returns: BOOL
// Qualifier: 根据主机创建目标服务器套接字，并连接
// Parameter: SOCKET * serverSocket
```

```
// Parameter: char * host
//************************************
BOOL ConnectToServer(SOCKET *serverSocket, char *host) {
	sockaddr_in serverAddr;
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(HTTP_PORT);
	//printf("%s", host);
	if (fobidden_flag == 1 && checkinf(host) == 1)
	{
		return FALSE;
	}
	HOSTENT *hostent = gethostbyname(host);
	if (!hostent) {
		printf("域名出错!!!");
		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) {
		printf("建立套接字出错!!!");
		return FALSE;
	}
	if (connect(*serverSocket, (SOCKADDR *)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
		printf("连接目标服务器出错!!!");
		closesocket(*serverSocket);
		return FALSE;
	}
	return TRUE;
}
int checkfishing(char* host)
{
	int equal = 1;
	for (int i = 0; i < max_fishing; i++)
	{
		if (strcmp(host, fishing_list[i].init_host) == 0)
		{
			memcpy(host, fishing_list[i].fishing_host, 50);//钓鱼
			return i;
		}
		int j = 0;
		equal = 1;
		while (host[j] != 0 || fishing_list[i].init_host[j] != 0)
```

```
            {
            if (host[j] != fishing_list[i].init_host[j])
            {
            equal = 0;
            break;
            }
            }
            if (equal == 1)
            {
            return i;
            }
        }
        return -1;
}
int checkuser(char* user)
{
        int i = 0;
        while (f_user[i] != NULL)
        {
            if (strstr(f_user[i], user) != NULL)
                return 1;
            i++;
        }
        return 0;
}
int checkinf(char* host)
{
        int i = 0;
        while (f_url[i] != NULL)
        {
            if (strstr(f_url[i], host)!=NULL)
                return 1;
            i++;
        }
        return 0;
}
void read_user(char path[])
{
        for (int i = 0; i < max_f; i++)
        {
            f_user[i] = NULL;
        }
        int i = 0;
        FILE *fp = fopen(path, "r");
```

```
    char Strurl[1024] = { 0 };
    if (fp == NULL)
    {
        printf("打开文件失败\n");
    }
    while (!feof(fp))
    {
        fgets(Strurl, 1024, fp);
        f_user[i] = new char[strlen(Strurl) + 1];
        ZeroMemory(f_user[i], strlen(Strurl) + 1);
        memcpy(f_user[i], Strurl, strlen(Strurl));
        printf("用户 IP 禁用：%s\n", f_user[i]);
        i++;
    }
    fclose(fp);
}
void read_void(char path[])
{
    for (int i = 0; i < max_f; i++)
    {
        f_url[i] = NULL;
    }
    int i = 0;
    FILE *fp = fopen(path, "r");
    char Strurl[1024] = { 0 };
    if (fp == NULL)
    {
        printf("打开文件失败\n");
    }
    while (!feof(fp))
    {
        fgets(Strurl, 1024, fp);
        f_url[i] = new char[strlen(Strurl) + 1];
        ZeroMemory(f_url[i], strlen(Strurl) + 1);
        memcpy(f_url[i], Strurl, strlen(Strurl));
        printf("网站禁用：%s", f_url[i]);
        i++;
    }
    fclose(fp);
}
void read_fishing(char path[])
{
    char ch;
    char fishing_host[1024] = { 0 };
```

```
    char fishing_head[1024] = { 0 };
    char init_host[1024] = { 0 };
    for (int i = 0; i < max_fishing; i++)
    {
        memset(fishing_list[i].fishing_head, 0, 500);
        memset(fishing_list[i].fishing_host, 0, 50);
        memset(fishing_list[i].init_host, 0, 50);
    }
    int j = 0;
    FILE *fp = fopen(path, "r");
    while (!feof(fp))
    {
        fgets(fishing_host, 1024, fp);
        printf("%s\n", fishing_host);
        memcpy(fishing_list[j].fishing_host, fishing_host, strlen(fishing_host) - 1);
        printf("%s\n", init_host);
        fgets(init_host, 1024, fp);
        memcpy(fishing_list[j].init_host, init_host, strlen(init_host) - 1);
        int k = 0;
        while ((ch = fgetc(fp)) != '!') {//请求报文以！结束
            if (ch == '\n')
            {
                fishing_head[k++] = '\r';
                fishing_head[k++] = '\n';
            }
            else
                fishing_head[k++] = ch;
        }
        memcpy(fishing_list[j].fishing_head, fishing_head, 500);
        if (ch == '!')
            break;

        j++;
    }
}
char* makechar(char* name)
{
    char* r_name = new char[100];
    memset(r_name, 0, 100);
    int i = 0;
    int i_name = 0;
    while (name[i] != NULL)
    {
        i_name = i_name + name[i];
```

```
            i++;
        }
        itoa(i_name, r_name, 10);
        return r_name;
}
/*int client_ret = 1, server_ret = 1;
char ch[2];
do{
server_ret = recv(((ProxyParam *)lpParameter)->serverSocket, ch, 1, 0);
client_ret = send(((ProxyParam *)lpParameter)->clientSocket, ch, server_ret, 0);
//printf("recv: %d, send: %d\n", server_ret, client_ret);
} while (client_ret >= 1 && server_ret >= 1);
*/
int checknum(char* buffer)
{
        char *p;
        char *ptr;
        const char * delim = "\r\n";
        char num[10] = { 0 }; return 200;
        p = strtok_s(buffer, delim, &ptr);//提取第一行
        memcpy(num, &p[9], 3);
        if (strcmp(num, "304") == 0)
            return 304;
        return 200;
}
int checkoutofdate(FILE* fp) {
        char date[40] = { 0 };
        long second = 0;
        long nowt = 0;
        struct tm *nowtime = (struct tm*)malloc(sizeof(struct tm));
        struct tm *time1 = (struct tm*)malloc(sizeof(struct tm));
        if (fp == NULL || 1)
        {
            return 0;
        }
        else
        {
            char strline[200] = { 0 };
            while (fp != NULL)
            {
                fgets(strline, 1024, fp);
                if (strline[0] == 'D'&&strline[1] == 'a'&&strline[2] == 't'&&strline[3] == 'e')
                {
                    memcpy(date, &strline[6], 25);
```

```
                timeconvert(date, time1);
                second = mktime(time1);
                break;
            }
        }
        if (nowt - second < age)
            return 0;
        else
            return 1;
    }
}
void addifsincemodified(char* buffer, HttpHeader* httpHeader) {
    char add[1024] = { 0 };
    memcpy(add, "if modified since", 20);
    strcat(add, httpHeader->date);
    strcat(buffer, add);
}

//比较周数，成功返回 0-6 的数，错误返回 7
//p 代表周数，取周数前 3 个字母，如 Mon 代表周 1，以此类推
//改动周几不影响返回的时间值，可以通过改动日期的日数来达到修改时间
int weekcmp(IN char *p)
{
    char week[7][5] = { "Sun\0", "Mon\0", "Tue\0", "Wed\0", "Thu\0", "Fri\0", "Sat\0" };
    int i;


    for (i = 0; i<7; i++)
        if (strcmp(p, week[i]) == 0)
            break;

    if (i == 7)
    {
        printf("fail to find week.\n");
        return i;
    }
    return i;
}
//比较月份，成功返回 0-11 的数，错误返回 12
//P 为月份的前三个字母，如 Feb 代表二月，以此类推
int monthcmp(IN char *p)
{
    char month[13][5] = { "Jan\0", "Feb\0", "Mar\0", "Apr\0", "May\0", "Jun\0", "Jul\0", "Aug\0",
"Sep\0", "Oct\0", "Nov\0", "Dec\0" };
```

```
    int i;
    for (i = 0; i<12; i++)
        if (strcmp(p, month[i]) == 0)
            break;
    if (i == 12)
    {
        printf("fail to find month.\n");
        return i;
    }
    return i;
}
//将字串格式的时间转换为结构体,返回距离 1970 年 1 月 1 日 0：0：0 的秒数，当字符串格式错误或超值时返回 0
//BUF 为类似 Tue May 15 14:46:02 2007 格式的，p 为时间结构体
time_t timeconvert(IN char *buf, OUT struct tm *p)
{

    char cweek[4];
    char cmonth[4];
    time_t second;

    sscanf(buf, "%s %s %d %d:%d:%d %d", cweek, cmonth, &(p->tm_mday), &(p->tm_hour), &(p->tm_min), &(p->tm_sec), &(p->tm_year));
    p->tm_year -= 1900;
    printf("****%s,%s*****\n", cweek, cmonth);
    p->tm_mon = monthcmp(cmonth);
    //改动周几不影响返回的时间值，可以通过改动日期的日数来达到修改时间
    p->tm_wday = weekcmp(cweek);
    if (p->tm_mon == 12 && p->tm_wday == 7)
    {
        printf("monthcmp() or weekcmp() fail to use.\n");
        return 0;
    }
    return second = mktime(p);
}
```