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	林燕燕		院系	人工智能		
班级	1903601		学号	190200501		
任课教师	李全龙		指导教师	李全龙		
实验地点	格物 207		实验时间	2021.10.30		
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)					
教师评语						

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

熟悉并掌握 Socket 网络编程的过程与技术；深入理解 HTTP 协议，掌握 HTTP 代理服务器的基本工作原理；掌握 HTTP 代理服务器设计与编程实现的基本技能。

实验内容：

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

(2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since头行），向原服务器确认缓存对象是否是最新版本。（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：

1. Socket 编程的客戶端和服务器端的主要步骤

a) socket 客戶端

明确目的服务器的 IP 地址、端口号以及传输层协议(TCP or UDP)，根据以上信息构造 socket用于通信；在接受消息后，关闭 socket 连接。

b) socket 服务器端

i. 对于 UDP 协议上的通信，无需提前建立连接，只需在开始时建立相应的 socket，进入无限循环，接收消息后直接与源地址进行通信即可。

ii. 对于 TCP 协议上的通信，服务器需要有一个 socket 负责控制，在进入无限循环前建立绑定指定的端口号，并在无限循环内，对于每一个连接新建 TCP 连接与源主机进行通信即可。

2. HTTP代理服务器原理

a) 首先初始化一个套接字，利用 blind() 函数将该套接字与服务器 host 地址绑定，地址设为 “127.0.0.1”；同时，也要绑定端口号，这里就按照指导书上的要求设置为“10240”。然后，利用 listen() 函数对该端口进行监听。

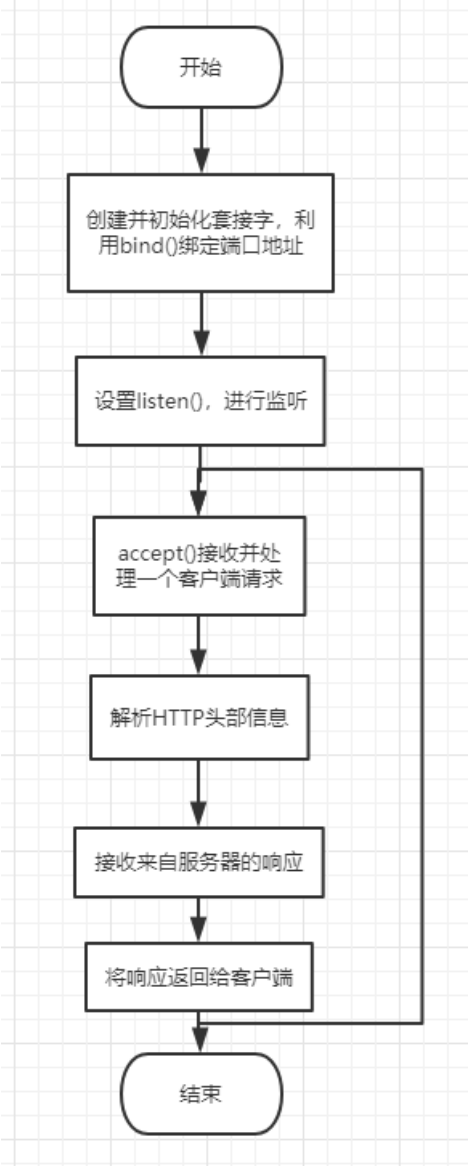
b) 通过设置 accept() 函数，对每个到来的请求进行接收和相应，为了提供效率，对每个请求都创建一个新的线程来处理。

c) 利用 recv() 和 send() 函数，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。在这里，代理服务器相当于一个中介，提供一个代理的服务，所有的请求和响应都经过它。

d) 处理完成后，等待 200 ms 后，关闭该线程，并清理缓存，然后继续接收并处理下一个请求。对于客户端而言，它只要将正常发送的请求发给代理服务器，就可以接收到对应

的响应。

3. HTTP代理服务器程序流程图



4. 实现 HTTP 代理服务器的关键技术及解决方案

e) 屏蔽网站

对请求过来的 HTTP 报文头部进行解析，提取出其中的访问地址 url ，检测其是否包含在要被屏蔽的网址列表中，如果包含，返回false

f) 屏蔽用户

识别访问代理服务器的客户端地址，若包含在被屏蔽用户列表中，则退出，继续监听。

g) 钓鱼网站

检测请求过来的 HTTP 报文头部，如果发现访问的网址是要被钓鱼的网址，则将该网址引导到其他网站（钓鱼网址），通过更改 HTTP 头部字段的 url （访问网址）、host 主机名和请求报文来实现。

h) cache 实现

i. 客户端第一次请求服务器中的数据时，代理服务器将该请求返回的响应缓存下来，

存到本地的文件下。

ii. 当客户端第二次访问该数据时，代理服务器检查本地是否有该请求的响应，如果没有，则继续缓存；如果有，则向服务器发送一个请求，该请求需要增加 “If-Modified-Since” 字段，通过此字段，告知服务器缓存资源最后修改的时间（可以将 “Date” 字段进行解析），服务器通过对比最后修改时间来判断缓存是否过期，如果没过期，服务器返回状态码304，代理服务器直接将本地缓存发送给客户端；如果缓存过期，服务器返回状态码200，同时返回一个更新过的响应，代理服务器接收后，将该响应发回给客户端，并更新本地缓存。

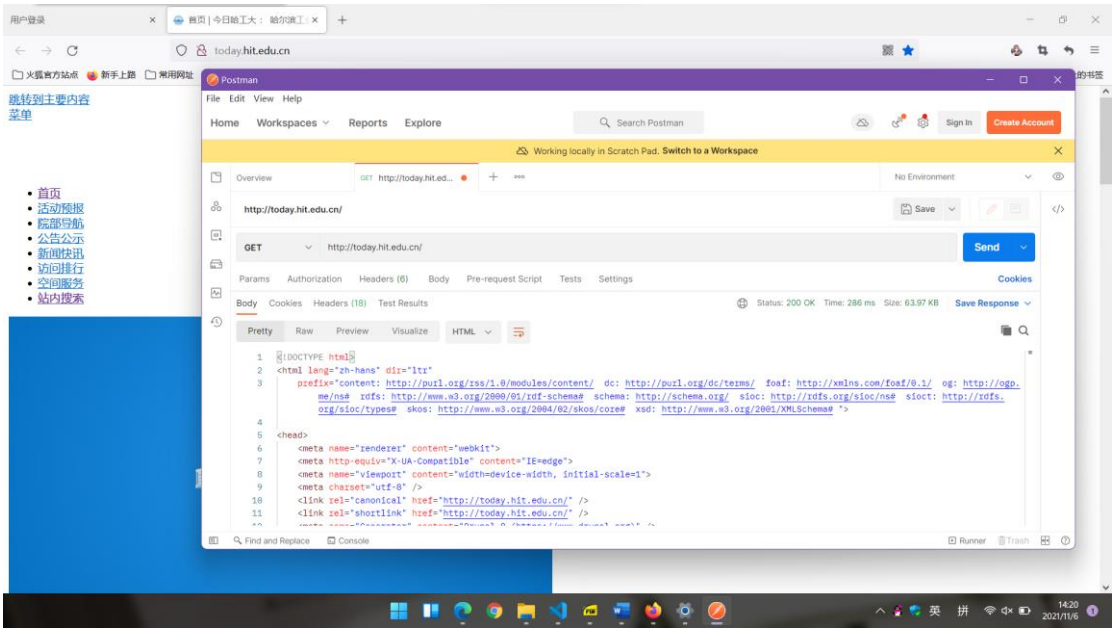
实验结果：

1. 基本功能

访问http:// today.hit.edu.cn/，访问的结果如下：

```
PS E:\Program\Network\Lab1\Code> g++ .\proxy.cpp -o .\proxy.exe -l ws2_32 ; .\proxy.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
```

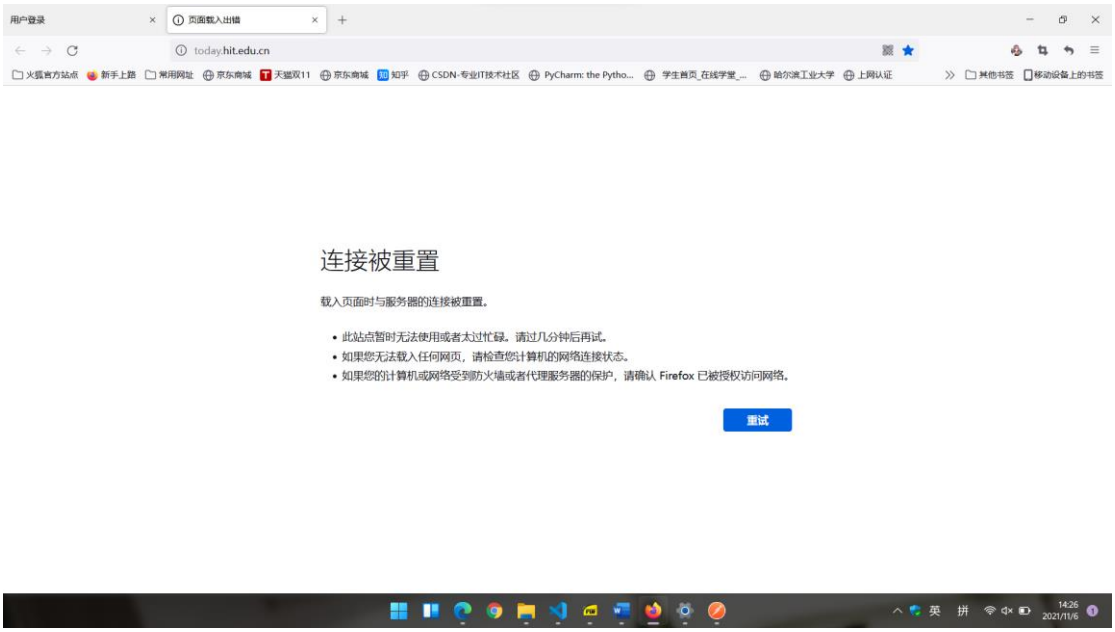
```
问题  输出  终端  调试控制台                                                    powershell

GET http://today.hit.edu.cn/ HTTP/1.1
http://today.hit.edu.cn/
代理连接主机 today.hit.edu.cn 成功

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 224633
Connection: keep-alive
Date: Sat, 06 Nov 2021 06:21:35 GMT
Server: Server
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
ETag: "36d79-5d017706c3701"
Accept-Ranges: bytes
Vary: Accept-Encoding
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
X-Boost-Cache: FULL
X-Varnish: 369495924
Age: 0
Via: 1.1 varnish-v4
X-Varnish-Cache: MISS
Accept-Ranges: bytes

<!DOCTYPE html>
<html lang="zh-hans" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/ dc: http://purl.org/dc/terms/ foaf: http://xmlns.com/foaf/0.1/ og: http://ogp.me/ns# rdfs: http://www.w3.org/2000/01/rdf-schema# schema: http://schema.org/ sioc: http://rdfs.org/sioc/ns# sioct: http://rdfs.org/sioc/types# skos: http://www.w3.org/2004/02/skos/core# xsd: http://www.w3.org/2001/XMLSchema# ">
  <head>
    <meta name="renderer" content="webkit">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta charset="utf-8" />
<link rel="canonical" href="http://today.hit.edu.cn/" />
<link rel="shortlink" href="http://today.hit.edu.cn/" />
<meta name="Generator" content="Drupal 8 (https://www.drupal.org)" />
<meta name="MobileOptimized" content="width" />
<meta name="HandheldFriendly" content="true" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```



2. 网站限制

对 http://today.hit.edu.cn/ 访问结果如下：

```
GET http://today.hit.edu.cn/ HTTP/1.1
http://today.hit.edu.cn/
====================Website blocked.====================
```



3. 用户限制

限制本地地址：

```
PS E:\Program\Network\Lab1\Code> g++ .\proxy.cpp -o .\proxy.exe -l ws2_32 ; .\proxy.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
==============User blocked.================
```

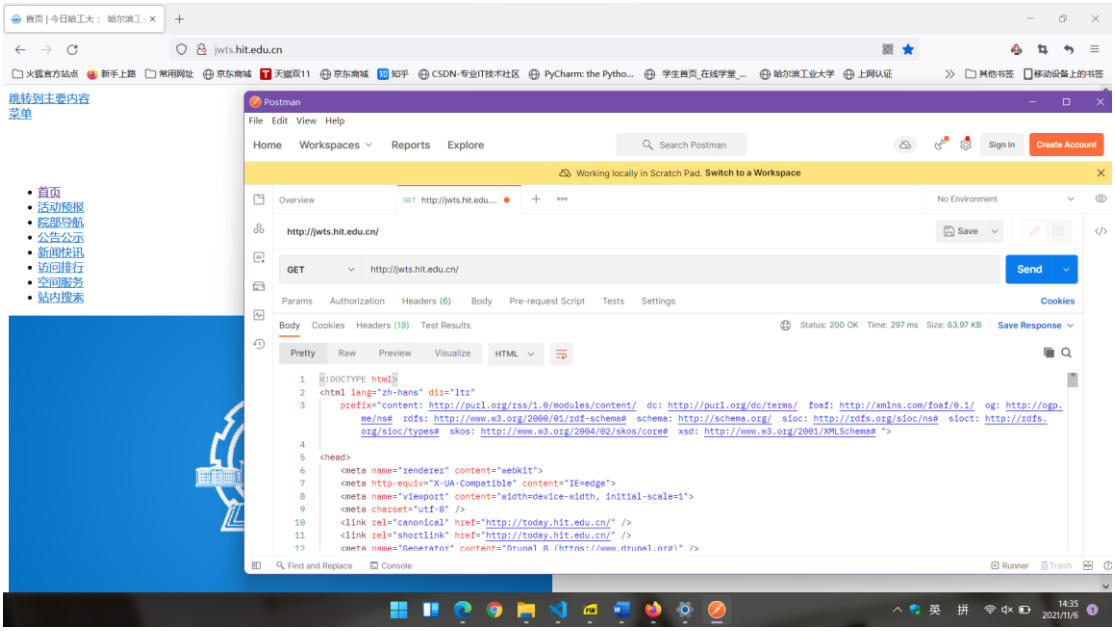
4. cache的实现

每次访问网址后将网页内容缓存到内存，下次访问时在请求报文中加入If-Modified-Since:字段，判断返回的响应码，若为304则使用本地缓存，若为200则使用返回报文，并更新缓存内容：

```
问题  输出  终端  调试控制台
Recieve a connection from 127.0.0.1:52715

GET http://today.hit.edu.cn/ HTTP/1.1
http://today.hit.edu.cn/
======================缓存存在======================
******************GET http://today.hit.edu.cn/ HTTP/1.1
User-Agent: PostmanRuntime/7.28.3
Accept: */*
Postman-Token: 522ba1fc-874d-4f44-911c-0b862b5cd47f
If-Modified-Since: Sat, 06 Nov 2021 06:32:37 GMT
Host: today.hit.edu.cn
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

**************代理连接主机 today.hit.edu.cn 成功

====================使用本地缓存====================
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 224633
Connection: keep-alive
Date: Sat, 06 Nov 2021 06:32:37 GMT
Server: Server
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
ETag: "36d79-5d017706c3701"
Accept-Ranges: bytes
Vary: Accept-Encoding
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
X-Boost-Cache: FULL
X-Varnish: 377791162
Age: 0
Via: 1.1 varnish-v4
X-Varnish-Cache: MISS
Accept-Ranges: bytes

<!DOCTYPE html>
<html  lang="zh-hans" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/
ogp.me/ns#  rdfs: http://www.w3.org/2000/01/rdf-schema#  schema: http://schema.org/  sioc:
```

5. 钓鱼网站

访问 http://jwts.hit.edu.cn/ ，被钓鱼至 http://today.hit.edu.cn/ ：



问题讨论：
参考代码中 goto 语句报错，将goto语句后定义的变量移至前面，在goto语句后定义新变量可能会出现问题； 设置缓存时需要获取时间，使用strtok_s()函数逐行提取报文，使用strstr()查找关键字是否存在，使用memcpy()函数复制内容； 钓鱼网站和确认缓存是否过期时需要修改报文内容，使用find()函数查找关键字位置，使用replace()函数替换数据。
心得体会：
经过此次实验，熟悉了Socket 网络编程，清楚客户端和服务器之间Socket通信过程；掌握了HTTP 代理服务器的基本工作原理；同时了解了钓鱼网站，禁止用户，禁止网站以及 Cache等的原理。还经过修改增加报文字段，对字符串相关函数的使用更加熟练。

源代码：

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>
#include <tchar.h>
#include <string>
using namespace std;
#pragma comment(lib, "Ws2_32.lib")
#define MAXSIZE 65507 // 发送数据报文的最大长度
#define HTTP_PORT 80   // http 服务器端口
// 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();
void ParseHttpHead(char *buffer, HttpHeader *httpHeader);
BOOL ConnectToServer(SOCKET *serverSocket, char *host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void Phishing(char *buffer, HttpHeader *httpHeader);
boolean ParseDate(char *buffer, char *tempDate);
```

```
void makeNewHTTP(char *buffer, char *Date);

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

//缓存相关参数
boolean haveCache = false;
struct Cache
{
	HttpHeader *htp;
	char buffer[MAXSIZE];
	char date[1024];
	Cache()
	{
		ZeroMemory(this->buffer, MAXSIZE);
		ZeroMemory(this->date, 1024);
		this->htp = new HttpHeader();
	}
};
Cache *cache[1024];
int cache_Pos;
//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;
};
#define NUM 100
char *Filter_website[NUM] = {}; // (char *)"today.hit.edu.cn"
char *Filter_users[NUM] = {(char *)"127.0.0.1"};     //
char *Phishing_website[NUM] = {(char *)"jwts.hit.edu.cn"};    //
char *Target_website[NUM] = {(char *)"today.hit.edu.cn"};

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;
	sockaddr_in remoteAddr;
	int nAddrlen = sizeof(remoteAddr);
	ProxyParam *lpProxyParam;
	HANDLE hThread;
	DWORD dwThreadID;
	//代理服务器不断监听
	while (true)
	{
		acceptSocket = accept(ProxyServer, (SOCKADDR *)&remoteAddr,
&nAddrlen);
		lpProxyParam = new ProxyParam;
		if (lpProxyParam == NULL)
		{
			continue;
		}
		////////////////////////////// 过滤用户
		BOOL blocked = false;
		for (int i = 0; i < NUM; i++)
		{
			if (Filter_users[i] == NULL)
			{
				break;
			}
			if (strcmp(inet_ntoa(remoteAddr.sin_addr), Filter_users[i])
== 0)
			{
				blocked = true;
				break;
			}
		}
		if (blocked)
		{
			printf("==============User blocked.================\n\n");
			continue;
		}
		////////////////////////////////
		lpProxyParam->clientSocket = acceptSocket;
```

```
            printf("Recieve a connection from %s:%d\n\n",
inet_ntoa(remoteAddr.sin_addr), remoteAddr.sin_port);
            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: 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);
	////////////////////////////////////////// 过滤网页
	for (int i = 0; i < NUM; i++)
	{
		if (Filter_website[i] == NULL)
		{
			break;
```

```
            }
            if (strcmp(host, Filter_website[i]) == 0)
            {
                  printf("====================Website
blocked.====================\n");
                  return FALSE;
            }
      }
      ////////////////////////////////////////
      if (!hostent)
      {
            return FALSE;
      }
      in_addr Inaddr = *((in_addr *)*hostent->h_addr_list);
      serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
      *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
      if (*serverSocket == INVALID_SOCKET)
      {
            return FALSE;
      }
      if (connect(*serverSocket, (SOCKADDR *)&serverAddr, sizeof(serverAddr))
== SOCKET_ERROR)
      {
            closesocket(*serverSocket);
            return FALSE;
      }
      return TRUE;
}
//**************************************
// 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)
{
      // GET http://today.hit.edu.cn/ HTTP/1.1
      // User-Agent: PostmanRuntime/7.28.3
      // Accept: */*
      // Postman-Token: fc56543e-1ed4-4a0e-8e6c-a01b278ea132
      // Host: today.hit.edu.cn
      // Accept-Encoding: gzip, deflate, br
```

```
	// Connection: keep-alive
	char Buffer[MAXSIZE];
	char *CacheBuffer;
	ZeroMemory(Buffer, MAXSIZE);
	SOCKADDR_IN clientAddr;
	int length = sizeof(SOCKADDR_IN);
	int recvSize;
	int ret;
	recvSize = recv(((ProxyParam *)lpParameter)->clientSocket, Buffer,
MAXSIZE, 0); // 接收报文
	HttpHeader *httpHeader = new HttpHeader();
	char *DateBuffer;
	char *field = (char *)"Date: ";
	char date[40]; // 缓存 date 字段
	char *p, *ptr, num[10], tempBuffer[MAXSIZE + 1];
	const char *delim = "\r\n";
	if (recvSize <= 0)
	{
		goto error;
	}
	CacheBuffer = new char[recvSize + 1];
	ZeroMemory(CacheBuffer, recvSize + 1);
	memcpy(CacheBuffer, Buffer, recvSize);
	// 解析 http 头部
	ParseHttpHead(CacheBuffer, httpHeader);
	delete CacheBuffer;
	////////////////////////////////// 钓鱼网站
	Phishing(Buffer, httpHeader);
	//////////////////////////////////
	////////////////////////////////////////////缓存 Cache

	for (int i = 0; i < 1024; i++)
	{ // 查找缓存
		if (cache[i] == NULL)
		{
			haveCache = false;
			cache_Pos = i; // 第一个空着的缓存位置
			break;
		}
		if (!strcmp(cache[i]->htp->method, httpHeader->method)
&& !strcmp(cache[i]->htp->url, httpHeader->url)
&& !strcmp(cache[i]->htp->host, httpHeader->host))
		{
```

```
                printf("=====================缓存存在
====================\n");
                haveCache = true;
                memcpy(date, cache[i]->date, 40);
                cache_Pos = i;
                makeNewHTTP(Buffer, date); // 修改请求报文
                printf("******************%s****************",Buffer);
                break;
            }
        }
        if (!ConnectToServer(&((ProxyParam
                                              *)lpParameter)
                                            ->serverSocket,
                                    httpHeader->host))
        {
            goto error;
        }
        printf("代理连接主机 %s 成功\n\n", httpHeader->host);
        //将客户端发送的 HTTP 数据报文直接转发给目标服务器
        ret = send(((ProxyParam *)lpParameter)->serverSocket, Buffer,
strlen(Buffer) + 1, 0);
        //等待目标服务器返回数据
        recvSize = recv(((ProxyParam
                                       *)lpParameter)
                                     ->serverSocket,
                             Buffer, MAXSIZE, 0);

        if (recvSize <= 0)
        {
            goto error;
        }
        // 查看是否更新
        ZeroMemory(num, 10);
        ZeroMemory(tempBuffer, MAXSIZE + 1);
        memcpy(tempBuffer, Buffer, strlen(Buffer));
        p = strtok_s(tempBuffer, delim, &ptr); //取报文第一行 ps. "HTTP/1.1 200
OK"
        memcpy(num, &p[9], 3);                          // 获取状态码
        if (strcmp(num, "304") == 0)
        { //状态码为 304 使用本地缓存
            printf("====================使用本地缓存====================\n");
            memcpy(Buffer, cache[cache_Pos]->buffer, MAXSIZE);
        }
        else if (strcmp(num, "200") == 0)
```

```
        { //状态码为 200 更新缓存
            if (haveCache == false)
            {
                cache[cache_Pos] = new Cache();
                memcpy(cache[cache_Pos]->htp->method, httpHeader->method,
strlen(httpHeader->method));
                memcpy(cache[cache_Pos]->htp->url, httpHeader->url,
strlen(httpHeader->url));
                memcpy(cache[cache_Pos]->htp->host, httpHeader->host,
strlen(httpHeader->host));
                memcpy(cache[cache_Pos]->htp->cookie, httpHeader->cookie,
strlen(httpHeader->cookie));
            }
            ParseDate(Buffer, date); // 获取时间
            memcpy(cache[cache_Pos]->date, date, strlen(date));
            memcpy(cache[cache_Pos]->buffer, Buffer, strlen(Buffer));
        }

        //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);
        printf("%s\n\n", Buffer); // 输出网站内容
//错误处理
error:
        printf("\n\n 关闭套接字.....\n\n");
        Sleep(200);
        closesocket(((ProxyParam *)lpParameter)->clientSocket);
        closesocket(((ProxyParam *)lpParameter)->serverSocket);
        _endthreadex(0);
        return 0;
}
//查找 HTTP 头部 "Date: "字段 获取时间
boolean ParseDate(char *buffer, char *tempDate)
{
        char *p, *ptr, temp[5];
        char *field = (char *)"Date: ";
        const char *delim = "\r\n";
        ZeroMemory(temp, 5);
        char tempbuffer[MAXSIZE];
        memcpy(tempbuffer, buffer, MAXSIZE);
        p = strtok_s(tempbuffer, delim, &ptr); // 获取一行
        int len = strlen(field);
```

```
	while (p)
	{ // 循环查找
		if (strstr(p, field) != NULL)
		{ // 在 buffer 中查找 "Date: " 字段  ps.Date: Thu, 28 Oct 2021 14:32:43 GMT
			memcpy(tempDate, &p[len], strlen(p) - len);
			return true;
		}
		p = strtok_s(NULL, delim, &ptr);
	}
	return false;
}

//修改 HTTP 请求报文
void makeNewHTTP(char *buffer, char *Date)
{
	char field[100] = "If-Modified-Since: ";
	strcat(field, Date);
	strcat(field, "\r\n");
	string line = buffer;
	int pos = line.find("Host");
	line.replace(pos, 0, field);
	for (int i = 0; i < line.size();i++){
			*buffer++ = line[i];
		}
}

//*************************************
// Method: Phishing
// FullName: Phishing
// Access: public
// Returns: void
// Qualifier: 修改 url,host,钓鱼
// Parameter: char * buffer
// Parameter: HttpHeader * httpHeader
//*************************************
void Phishing(char *buffer, HttpHeader *httpHeader)
{
	for (int i = 0; i < NUM; i++)
	{
		if (Phishing_website[i] == NULL || Target_website[i] == NULL)
		{
			break;
		}
```

```
            if (strcmp(Phishing_website[i], httpHeader->host) == 0)
            {
                  char url[1024] = "http://";
                  strcat(url, Target_website[i]);
                  strcat(url, "/");
                  memcpy(httpHeader->url, url, strlen(Target_website[i]));
                  memcpy(httpHeader->host, Target_website[i],
strlen(Target_website[i]));
                  string line = buffer;
                  // GET http://
                  int pos = line.find("Get") + 12;
                  int len = strlen(Phishing_website[i]);
                  line.replace(pos, len, Target_website[i]);
                  pos = line.find("Host: ") + 6;
                  len = strlen(Phishing_website[i]);
                  line.replace(pos, len, Target_website[i]);
                  for (int i = 0; i < line.size();i++){
                        *buffer++ = line[i];
                  }
                  break;
            }
      }
}
//*************************************
// 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);
	}
}
```