

**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 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 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    1. 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。   * 1. socket服务器端      1. 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。      2. 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。  1. HTTP代理服务器原理    1. 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。    2. 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。    3. 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。    4. 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。 2. HTTP代理服务器程序流程图      1. 实现 HTTP 代理服务器的关键技术及解决方案    1. 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   * 1. 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。   * 1. 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。   * 1. cache 实现      1. 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。      2. 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：         1. ⽹站限制   对 http://today.hit.edu.cn/ 访问结果如下：       1. ⽤⼾限制   限制本地地址：     1. 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：     1. 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);

    }

}