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	徐柯炎		院系	计算机科学与技术学院	
班级	2103602		学号	2021110683	
任课教师	刘亚维		指导教师	刘亚维	
实验地点	格物 207		实验时间	2023.10.21	
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					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

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：

必做部分：

1. 主函数：

```
int main(int argc, char* argv[])
{
    sockaddr_in addr_in;
    int addr_len = sizeof(SOCKADDR);

    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) {
        //获取用户主机ip地址
        acceptSocket = accept(ProxyServer, (SOCKADDR*)&addr_in, &(addr_len));
        lpProxyParam = new ProxyParam;
        if (lpProxyParam == NULL) {
            continue;
        }
```

主函数部分首先进行socket的初始化，接着代理服务器开始不断监听，也就是不断获取主机ip地址，如果得到了ip地址，那就开一个线程为用户主机进行网页访问的服务。

2. 初始化socket：

a) 初始化套接字库

如下图所示，这一部分的主要实现的是初始化代理服务器的 Socket。如果初始化成功返回 TRUE。将本地的端点地址绑定到套接字上。

```
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;
}
```

b） 创建 Socket

如下图所示，利用 socket(AF_INET,SOCK_STREAM, 0)方法创建套接字，第一个参数代表协议族，AF_INET 表示是 Internet 通信;第二个参数代表套接字类型，SOCK_STREAM 表示是面向 TCP 连接的流式套接字；第三个参数代表协议号，默认设置为 0；

```
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;
}
```

3. 创建线程函数：

如下图所示，在线程函数中，首先需要初始化缓存，然后接收客户端请求消息，从客户端获得 http 报文，调用 recv()函数接受客户端请求的消息，函数返回值为实际收到的消息字节数，消息内容缓存在 Buffer 中。然后将 Buffer 拷贝一份到 CacheBuffer 中，然后用拷贝的这一份缓存开始解析 http 头部。得到的信息就是 http 请求报文的头部。

```
//从客户端获得http报文，存入Buffer
recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
if (recvSize <= 0) {
    goto error;
}

CacheBuffer = new char[recvSize + 1];
ZeroMemory(CacheBuffer, recvSize + 1);
memcpy(CacheBuffer, Buffer, recvSize);  //拷贝buffer
//connect方式不创建连接
if (!ParseHttpHead(CacheBuffer, httpHeader)) {
    goto error;
}
delete CacheBuffer;
```

之后的工作就是是否能连接到我们需要访问的网址，如下图所示，首先调用 ConnectToServer 函数来和服务器进行连接，下一步就是将客户端发送的 HTTP 请求报文转发给目标服务器，在这里我们通过 send 函数将存在 buffer 缓存里的报文发送给目标服务器，然后通过 recv 函数等待目标服务器返回数据报，最后通过 send 函数将目标服务器返回的数据报发送给客户端，实现代理服务器的功能。

```
if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket, httpHeader->host)) {
    goto error;
}
printf("代理连接主机 %s 成功\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;
}
```

最后是异常处理，如果在过程中有异常均跳转到 error，结束线程运行。

```
    //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;
```

4. 解析http头部：

主要通过ParseHttpHead函数来实现。

```
p = strtok_s(buffer, delim, &ptr);//提取第一行
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);
}
else {
    return false;
}
//打印url
printf("url = %s\n", httpHeader->url);
p = strtok_s(NULL, delim, &ptr);
```

如上图所示，首先提取报文首部的第一行，得知该报文是get方式还是post方式的。

```
while (p) {
    //printf("%s\n", 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);
}
```

接着循环提取报文，提取http首部的其他必要信息。

5. 建立连接：

这一部分主要通过ConnectToServer函数来实现，这个函数会根据发送端套接字的协议族和端口号还有套接字类型，以及目的主机的IP地址和端口号进行建立连接，如果连接成功，放回TRUE。

```
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;
    }
    in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
    serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
    //printf("%s\n", 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;
}
```

(局部变量) sockaddr_in serverAddr
联机搜索

选做部分：

1. 支持cache功能：

首先我们使用makeflienname函数，通过网站的url来为缓存文件命名。如果这个文件存在，则在给服务器发送的报文中添加If-Modified-Since字段（通过makeNewHttp函数），询问服务器该网站的最后修改时间，并将havecache设置为true；接着将报文发送给服务器，根据服务器返回的报文来进行接下来的操作。

```
FILE* fp;
errno_t err;
makeFilename(httpHeader->url, filename);
err = fopen_s(&in, filename, "rb");
//如果有缓存文件，插入If-Modified-Since字段
if (in != NULL)
{
    fread(fileBuffer, sizeof(char), MAXSIZE, in);
    fclose(in);
    getDate(fileBuffer, field, date_str);
    printf("插入If-Modified-Since: %s\n", date_str);
    makeNewHTTP(Buffer, date_str);
    haveCache = true;
}
```

如下图所示，如果有缓存的话进入getcache函数来进行接下来的操作，如果没有缓存的话就缓存报文。

```
// 是否有缓存，没有缓存报文
if (haveCache)
{
    getCache(Buffer, filename);
}
else {
    makeCache(Buffer, httpHeader->url);  //缓存报文
}
```

Getcache函数如下图所示。首先将buffer复制一份到tempbuffer中，然后通过tempbuffer来获取服务器报文返回的状态码。如果状态码为304，就把本机上缓存的文件读入buffer中再发送回主机，如果状态码为200，则不对buffer进行操作，将服务器传回来的报文缓存一遍后再发送给客户端。

```
void getCache(char* buffer, char* filename) {
    char* p, * ptr, num[10], tempBuffer[MAXSIZE + 1];
    const char* delim = "\r\n";
    errno_t err;
    ZeroMemory(num, 10);
    ZeroMemory(tempBuffer, MAXSIZE + 1);

    //对tempbuffer进行操作
    memcpy(tempBuffer, buffer, strlen(buffer));

    p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
    memcpy(num, &p[9], 3);
    //printf("%s\n", buffer);

    if (strcmp(num, "304") == 0) {  //主机返回的报文中的状态码为304时返回已缓存的内容
        ZeroMemory(buffer, strlen(buffer));
        printf("*************************************\n");
        printf("从本机获得缓存\n");
        FILE* in = NULL;
        err = fopen_s(&in, filename, "r");
        //从文件中获取缓存的内容
        if (in != NULL) {
            fread(buffer, sizeof(char), MAXSIZE, in);
            fclose(in);
        }
    }
```

Makecache部分如下图所示，首先还是将buffer复制一份到tempbuffer中，然后通过tempbuffer来获取服务器报文返回的状态码。如果状态码为200，即成功访问，则构建cache文件，将buffer中存放的报文拷贝一份放到文件中；如果状态码为404，说明访问失败，这时不构建文件。

```
p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
memcpy(num, &p[9], 3);
//printf("%s\n", num);

if (strcmp(num, "200") == 0) {  //状态码是200时缓存
    // 200指成功访问，404就是没成功

    // 构建文件
    char filename[100];
    ZeroMemory(filename, 100);
    makeFilename(url, filename);
    printf("filename : %s\n", filename);

    FILE *out, *fp;
    errno_t err;
    err = fopen_s(&fp, filename, "w");
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);
    printf("**************************************\n");
    printf("网页已经被缓存\n");
}
```

2. 网站过滤：

实现这个功能的代码在 ProxyThread 函数中，如下图所示。这个功能很简单，就是在一开始得到目的网址的 url 后，检查是否和我们想要屏蔽的网址是否相同，如果相同，直接跳转到 error。

```
#define invalid_website "http://today.hit.edu.cn/" //屏蔽网址

if (strcmp(httpHeader->url, invalid_website) == 0)
{
    printf("***********该网站已被屏蔽***********\n");
    goto error;
}
```

3. 用户过滤：

实现这个功能的代码在主函数中，如下图所示。这一部分是通过 accept 函数得到我们客户机的 IP 地址，进行匹配，如果和我们需要屏蔽的 IP 地址，就直接用户过滤。

```
const int ProxyPort = 10240;
const char* forbid_user[10] = { "127.0.0.1" };  //被屏蔽的用户IP

//用户过滤
if (!strcmp(forbid_user[0], inet_ntoa(addr_in.sin_addr)) && flag)//inet_ntoa把网络号
{
    printf("该用户访问受限\n");
    continue;
}
```

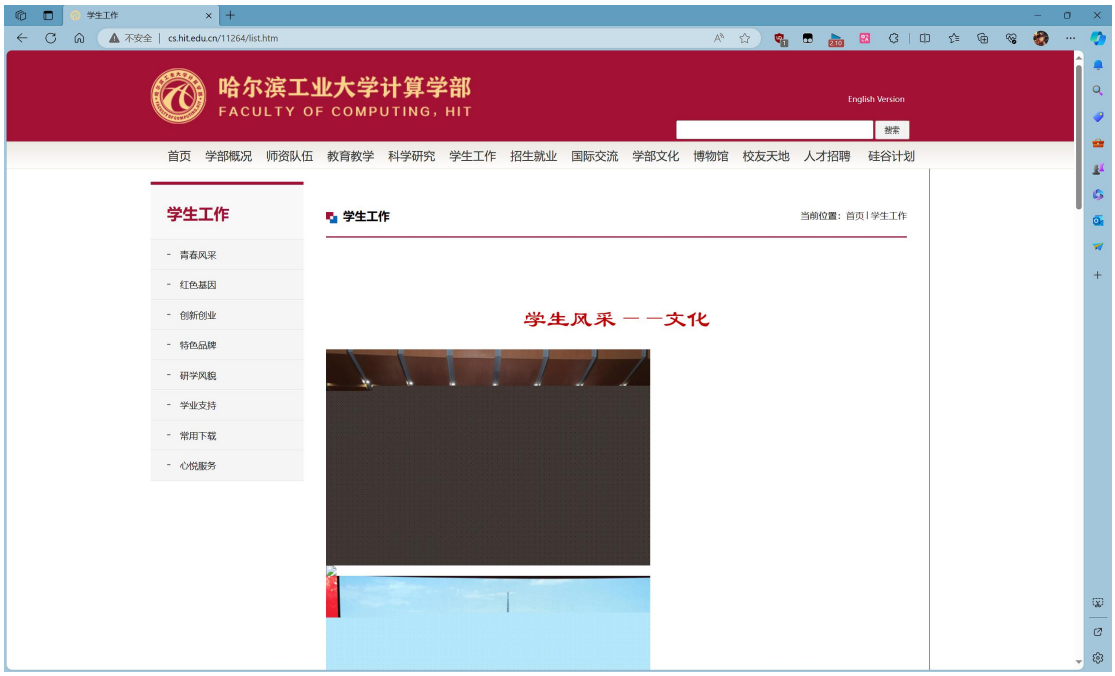
4. 网站引导：

实现这个功能的代码也在 ProxyThread 函数中，如下图所示。实现钓鱼也非常简单，匹配到和钓鱼原网址相同的网址，我们修改 http 头部的目的网址的 ip 地址和端口号，修改目的主机名，然后由代理服务器发送修改后的请求报文。

```
#define fish_web_src "http://www.7k7k.com/" //钓鱼源网址
#define fish_web_url "http://jwts.hit.edu.cn/" //钓鱼目的网址
#define fish_web_host "jwts.hit.edu.cn" //钓鱼目的地址的主机名

if (strcmp(httpHeader->url, fish_web_src) == 0)
{
    printf("*******目标网址已被引导*******\n");
    memcpy(httpHeader->host, fish_web_host, strlen(fish_web_host) + 1);
    memcpy(httpHeader->url, fish_web_url, strlen(fish_web_url));
}
```

实验结果：

1. 实现代理服务器功能，访问计算学部官网



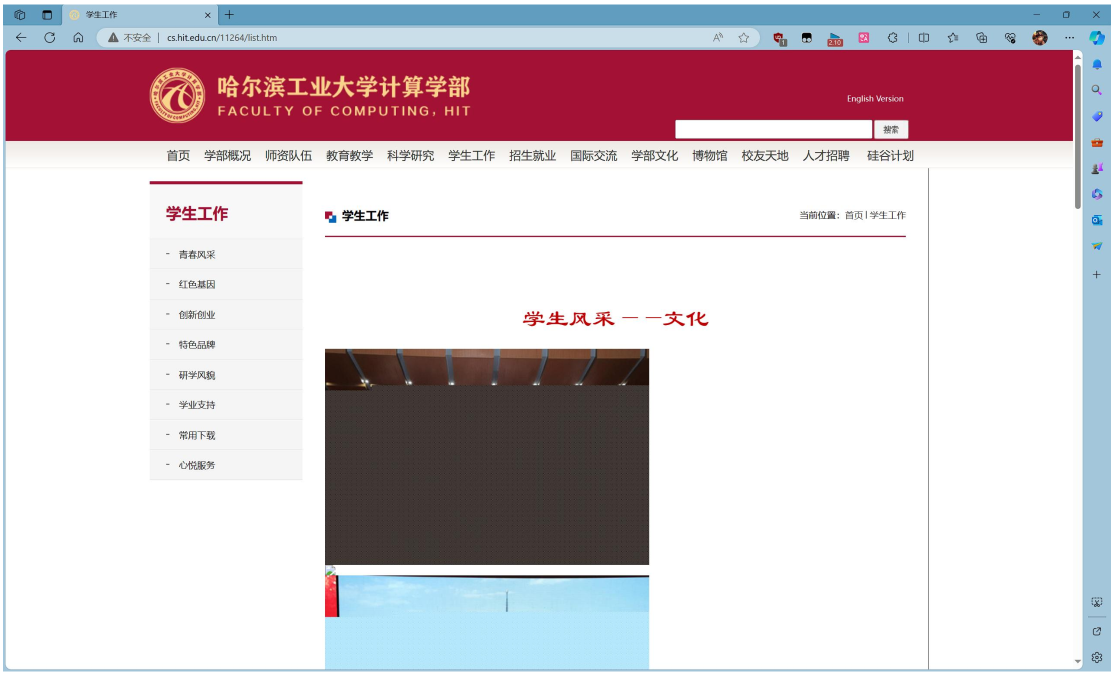

```
************************************
网页已经被缓存
数据已经发送给用户

关闭套接字
url = http://cs.hit.edu.cn/_upload/article/images/44/ae/64b3741c4cfbb6e330cedda94ae8/9ec2e4c3-d70f-4a70-a7a9-90077d2bd44
6.jpg
代理连接主机 cs.hit.edu.cn 成功
filename : httpcshiteducnuploadarticleimages44ae64b3741c4cfbb6e330cedda94ae89ec2e4c3d70f4a70a7a990077d2bd4.txt
************************************
网页已经被缓存
数据已经发送给用户

关闭套接字
url = http://cs.hit.edu.cn/_upload/article/images/44/ae/64b3741c4cfbb6e330cedda94ae8/40fb2a28-566f-46c6-a287-892b8a982eb
9.jpg
代理连接主机 cs.hit.edu.cn 成功
filename : httpcshiteducnuploadarticleimages44ae64b3741c4cfbb6e330cedda94ae840fb2a28566f46c6a287892b8a982e.txt
************************************
网页已经被缓存
数据已经发送给用户

关闭套接字
关闭套接字
url = http://cs.hit.edu.cn/_upload/article/images/44/ae/64b3741c4cfbb6e330cedda94ae8/666bac0a-71da-4136-81a9-5301d6e841e
0.jpg
代理连接主机 cs.hit.edu.cn 成功
filename : httpcshiteducnuploadarticleimages44ae64b3741c4cfbb6e330cedda94ae8666bac0a71da413681a95301d6e841.txt
************************************
网页已经被缓存
```

2. 实现缓存，并更新缓存文件

缓存文件如下图所示：

httpcshiteducn11264listhtm.txt	2023/10/21 15:58	文本文档
httpcshiteducnjsjqueryui1121jqueryuiminjs...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档

更新缓存，再次登录官网，可以看到缓存已经更新成功。

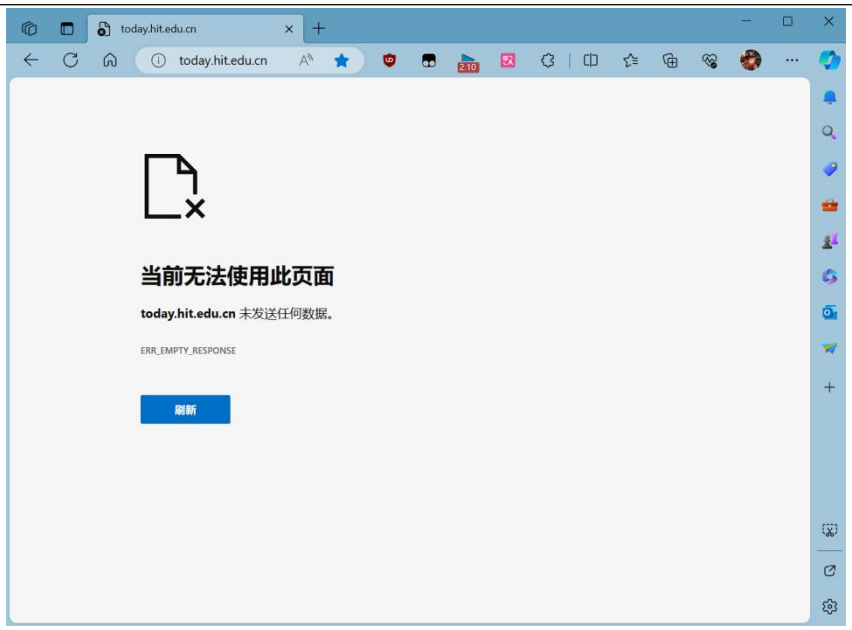


更新后的缓存文件：

httpcshiteducn11264listhtm.txt	2023/10/21 15:58	文本文档
httpcshiteducnjsjqueryui1121jqueryuiminjs...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档
httpcshiteducnuploadarticleimages44ae64...	2023/10/21 15:58	文本文档

3. 屏蔽网站今日哈工大

可以看到提示，该网站已被屏蔽。




```
关闭套接字
url = http://today.hit.edu.cn/
**********该网站已被屏蔽**********
关闭套接字
关闭套接字
```

4. 屏蔽特定用户

本次屏蔽用户的主机是127.0.0.1。可以看到提示该用户已被过滤：

```
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
该用户访问受限
该用户访问受限
该用户访问受限
```

5. 实现钓鱼，钓鱼源网址是7k7k小游戏，钓鱼后是哈工大教务处网站


```
url = http://www.7k7k.com/
*******目标网址已被引导*******
代理连接主机 jwts.hit.edu.cn 成功
filename : httpjwtshiteducn.txt
****************************************
网页已经被缓存
数据已经发送给用户
```

问题讨论：

(1) 实验指导书给出的源码中#pragma comment(lib,"Ws2_32.lib") 编译失败，查询资料得知由于使用的是 VSCode 中的 MinGW 编译器，该编译器不支持这种写法，于是换用 VS，成功解决这一问题。

(2) 一开始不知道如何获取客户端的 ip 地址，通过查找资料得知，使用 accept 函数可获得客户端主机的 ip 地址。

(3) fopen 函数在编译的时候会报错，原因是 fopen 不是安全函数。通过查阅资料换成了更加安全的 fopen_s 函数。

(4) strcat 函数在编译的时候会报错，原因是 strcat 不是安全函数。通过查阅资料换成了更加安全的 strcat_s 函数。

(5) 缓存文件命名时使用原 url 对文件命名会产生错误，于是在命名时把 url 中的‘/’去掉就不会产生报错。

(6) Socket编程的客户端和服务端主要步骤

a) 客户端

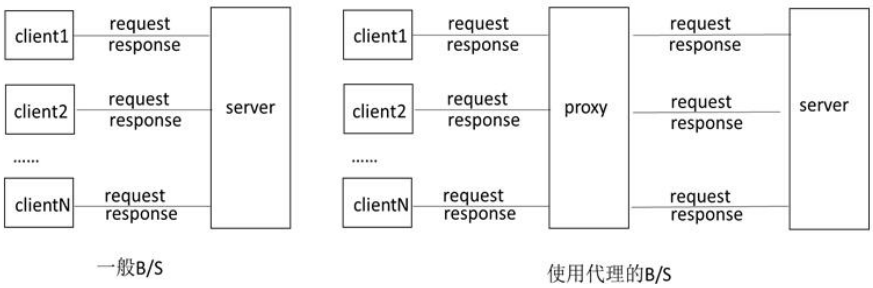
- 初始化套接字库
- 创建Socket
- 向服务器发出连接请求
- 连接建立后，向服务器请求数据，并置于等待状态，等待服务器返回数据
- 关闭连接
- 关闭套接字库

b) 服务端

- 初始化套接字库
- 创建套接字
- 绑定套接字
- 监听端口
- 接受连接请求，返回新的套接字
- 接受客户端请求消息，返回请求数据，与其通信
- 关闭套接字
- 关闭套接字库

(7) HTTP代理服务器原理

代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。如下图所示，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。

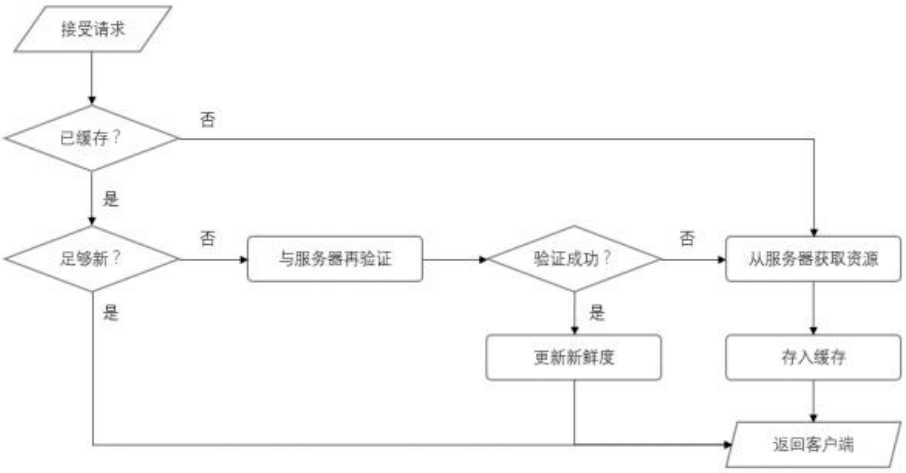


Web应用通信方式对比

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。

(8) HTTP代理服务器流程图



(9) 实现HTTP代理服务器的关键技术及解决方案

a) 单用户代理服务器

单用户的简单代理服务器可以设计为一个非并发的循环服务器。首先，代理服务器创建HTTP代理服务的TCP主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，读取客户端的HTTP请求报文，通过请求行中的URL，解析客户期望访问的原服务器IP地址；创建访问原（目标）服务器的TCP套接字，将HTTP请求报文转发给目标服务器，接收目标服务器的响应报文，当收到响应报文之后，将响应报文转发给客户端，最后关闭套接字，等待下一次连接。

b) 多用户代理服务器

多用户的简单代理服务器可以实现为一个多线程并发服务器。首先，代理服务器创建HTTP代理服务的TCP主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，创建一个子线程，由子线程执行上述一对一的代理过程，服务结束之后子线程终止。与此同时，主线程继续接受下一个客户的代理服务。

心得体会：

1. 本次对 HTTP 代理服务器的实现，更加理解了代理服务器的原理和执行过程；
2. 学习了多线程编程相关的函数，以及多线程安全的函数，同时感受到了多线程执行的好处；
3. 同时学会了 socket 编程，更加清晰了 socket 库中实现 HTTP 连接的各类函数及其作用；
4. 对用户过滤、网站过滤、cache 缓存以及网站引导的基本原理有了一个大概的了解，并基本掌握了实现的方法

附件：lab1 源代码

```
1. #include <stdio.h>
2. #include <Windows.h>
3. #include <process.h>
```

```
#include <string.h>

#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

#define invalid_website "http://today.hit.edu.cn/" //屏蔽网址
#define fish_web_src "http://www.7k7k.com/" //钓鱼源网址
#define fish_web_url "http://jwts.hit.edu.cn/" //钓鱼目的网址
#define fish_web_host "jwts.hit.edu.cn" //钓鱼目的地址的主机名

//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);
BOOL ConnectToServer(SOCKET* serverSocket, char* host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void makeFilename(char* url, char* filename);
void getCache(char* buffer, char* filename);
void makeCache(char* buffer, char* url);
void makeNewHTTP(char* buffer, char* value);
void getDate(char* buffer, char* field, char* tempDate);

//代理相关参数
SOCKET ProxyServer;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 10240;
const char* forbid_user[10] = { "127.0.0.1" };  //被屏蔽的用户 IP
bool flag = true;  //用户过滤开启或关闭

//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};
```

```
47. //DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};
48. 
49. struct ProxyParam {
50.     SOCKET clientSocket;
51.     SOCKET serverSocket;
52. };
53. 
54. int main(int argc, char* argv[])
55. {
56.     sockaddr_in addr_in;
57.     int addr_len = sizeof(SOCKADDR);
58. 
59.     printf("代理服务器正在启动\n");
60.     printf("初始化...\n");
61.     if (!InitSocket()) {
62.         printf("socket 初始化失败\n");
63.         return -1;
64.     }
65.     printf("代理服务器正在运行，监听端口 %d\n", ProxyPort);
66.     SOCKET acceptSocket = INVALID_SOCKET;
67.     ProxyParam* lpProxyParam;
68.     HANDLE hThread;
69.     DWORD dwThreadID;
70.     //代理服务器不断监听
71.     while (true) {
72.         //获取用户主机 ip 地址
73.         acceptSocket = accept(ProxyServer, (SOCKADDR*)&addr_in, &(ad
    dr_len));
74.         lpProxyParam = new ProxyParam;
75.         if (lpProxyParam == NULL) {
76.             continue;
77.         }
78.         //用户过滤
79.         if (!strcmp(forbid_user[0], inet_ntoa(addr_in.sin_addr)) &&
    flag)//inet_ntoa 把网络字节序的地址转化为点分十进制的地址
80.         {
81.             printf("该用户访问受限\n");
82.             continue;
83.         }
84.         lpProxyParam->clientSocket = acceptSocket;
85.         hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread, (LPV
    OID)lpProxyParam, 0, 0);
86.         CloseHandle(hThread);
87.         Sleep(200);
```

```
88.     }
89.     closesocket(ProxyServer);
90.     WSACleanup();
91.     return 0;
92. }
93.
94. //************************************
95. // Method:  InitSocket
96. // FullName: InitSocket
97. // Access:public
98. // Returns:BOOL
99. // Qualifier: 初始化套接字
100. //************************************
101. BOOL InitSocket() {
102.     //加载套接字库（必须）
103.     WORD wVersionRequested;
104.     WSADATA wsaData;
105.     //套接字加载时错误提示
106.     int err;
107.     //版本 2.2
108.     wVersionRequested = MAKEWORD(2, 2);
109.     //加载 dll 文件 Scoket 库
110.     err = WSAStartup(wVersionRequested, &wsaData);
111.     if (err != 0) {
112.         //找不到 winsock.dll
113.                     printf(" 加 载   winsock  失 败 ， 错 误 代 码 为: %d\n", WSAGetLastError());
114.         return FALSE;
115.     }
116.     if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
117.     {
118.         printf("不能找到正确的 winsock 版本\n");
119.         WSACleanup();
120.         return FALSE;
121.     }
122.     ProxyServer = socket(AF_INET, SOCK_STREAM, 0);
123.     if (INVALID_SOCKET == ProxyServer) {
124.                     printf(" 创 建 套 接 字 失 败 ， 错 误 代 码 为: %d\n", WSAGetLastError());
125.         return FALSE;
126.     }
127.     ProxyServerAddr.sin_family = AF_INET;
128.     ProxyServerAddr.sin_port = htons(ProxyPort);
```

```
129.     ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
130.     if (bind(ProxyServer, (SOCKADDR*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
131.         printf("绑定套接字失败\n");
132.         return FALSE;
133.     }
134.     if (listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR) {
135.         printf("监听端口%d 失败", ProxyPort);
136.         return FALSE;
137.     }
138.     return TRUE;
139. }
140. 
141. //************************************
142. // Method:ProxyThread
143. // FullName: ProxyThread
144. // Access:public
145. // Returns:unsigned int __stdcall
146. // Qualifier: 线程执行函数
147. // Parameter: LPVOID lpParameter
148. //************************************
149. unsigned int __stdcall ProxyThread(LPVOID lpParameter) {
150.     char Buffer[MAXSIZE];   //缓存
151.     char* CacheBuffer;      //缓存指针
152.     ZeroMemory(Buffer, MAXSIZE);
153. 
154.     SOCKADDR_IN clientAddr;
155.     int length = sizeof(SOCKADDR_IN);
156. 
157.     char fileBuffer[MAXSIZE];
158.     char *filename = (char*)calloc(100, sizeof(char));
159.     FILE* in;
160. 
161.     char* field = (char*)"Date";
162.     char date_str[30];
163.     ZeroMemory(date_str, 30);
164. 
165.     BOOL haveCache = false;
166.     HttpHeader* httpHeader = new HttpHeader();
167. 
168.     int recvSize;
169.     int ret;
170. 
171.     //从客户端获得 http 报文，存入 Buffer
```

```
172.     recvSize = recv(((ProxyParam*)lpParameter)->clientSocket, Buffe
    r, MAXSIZE, 0);
173.     if (recvSize <= 0) {
174.         goto error;
175.     }
176.
177.     CacheBuffer = new char[recvSize + 1];
178.     ZeroMemory(CacheBuffer, recvSize + 1);
179.     memcpy(CacheBuffer, Buffer, recvSize);  //拷贝 buffer
180.     //connect 方式不创建连接
181.     if (!ParseHttpHead(CacheBuffer, httpHeader)) {
182.         goto error;
183.     }
184.     delete CacheBuffer;
185.
186.     FILE* fp;
187.     errno_t err;
188.     makeFilename(httpHeader->url, filename);
189.     err = fopen_s(&in, filename, "rb");
190.     //如果有缓存文件，插入 If-Modified-Since 字段
191.     if (in != NULL)
192.     {
193.         fread(fileBuffer, sizeof(char), MAXSIZE, in);
194.         fclose(in);
195.         getDate(fileBuffer, field, date_str);
196.         printf("插入 If-Modified-Since: %s\n", date_str);
197.         makeNewHTTP(Buffer, date_str);
198.         haveCache = true;
199.     }
200.
201.     if (strcmp(httpHeader->url, invalid_website) == 0)
202.     {
203.         printf("**********该网站已被屏蔽**********\n");
204.         goto error;
205.     }
206.
207.     if (strcmp(httpHeader->url, fish_web_src) == 0)
208.     {
209.         printf("*******目标网址已被引导*******\n");
210.         memcpy(httpHeader->host, fish_web_host, strlen(fish_web_hos
    t) + 1);
211.         memcpy(httpHeader->url, fish_web_url, strlen(fish_web_url));

212.     }
```

```
213.
214.     if (!ConnectToServer(&((ProxyParam*)lpParameter)->serverSocket,
     httpHeader->host)) {
215.         goto error;
216.     }
217.     printf("代理连接主机 %s 成功\n", httpHeader->host);
218.
219.     //将客户端发送的 HTTP 数据报文直接转发给目标服务器
220.     ret = send(((ProxyParam*)lpParameter)->serverSocket, Buffer, st
    rlen(Buffer) + 1, 0);
221.     //等待目标服务器返回数据
222.     recvSize = recv(((ProxyParam*)lpParameter)->serverSocket, Buffe
    r, MAXSIZE, 0);
223.     if (recvSize <= 0) {
224.         goto error;
225.     }
226.
227.     // 是否有缓存，没有缓存报文
228.     if (haveCache)
229.     {
230.         getCache(Buffer, filename);
231.     }
232.     else {
233.         makeCache(Buffer, httpHeader->url);  //缓存报文
234.     }
235.
236.     //将目标服务器返回的数据直接转发给客户端
237.     ret = send(((ProxyParam*)lpParameter)->clientSocket, Buffer, si
    zeof(Buffer), 0);
238.     printf("数据已经发送给用户\n\n");
239.
240.     //错误处理
241. error:
242.     printf("关闭套接字\n");
243.     Sleep(200);
244.     closesocket(((ProxyParam*)lpParameter)->clientSocket);
245.     closesocket(((ProxyParam*)lpParameter)->serverSocket);
246.     delete lpParameter;
247.     _endthreadex(0);
248.     return 0;
249. }
250.
251. //************************************
252. // Method:ParseHttpHead
```

```
253. // FullName: ParseHttpHead
254. // Access:public
255. // Returns:void
256. // Qualifier: 解析 TCP 报文中的 HTTP 头部
257. // Parameter: char * buffer
258. // Parameter: HttpHeader * httpHeader
259. //**********************************
260. int ParseHttpHead(char* buffer, HttpHeader* httpHeader) {
261.     char* p;
262.     char* ptr;
263.     const char* delim = "\r\n";
264. 
265.     p = strtok_s(buffer, delim, &ptr);//提取第一行
266.     if (p[0] == 'G') {//GET 方式
267.         memcpy(httpHeader->method, "GET", 3);
268.         memcpy(httpHeader->url, &p[4], strlen(p) - 13);
269.     }
270.     else if (p[0] == 'P') {//POST 方式
271.         memcpy(httpHeader->method, "POST", 4);
272.         memcpy(httpHeader->url, &p[5], strlen(p) - 14);
273.     }
274.     else {
275.         return false;
276.     }
277.     //打印 url
278.     printf("url = %s\n", httpHeader->url);
279.     p = strtok_s(NULL, delim, &ptr);
280.     while (p) {
281.         //printf("%s\n", p);
282.         switch (p[0]) {
283.         case 'H'://Host
284.             memcpy(httpHeader->host, &p[6], strlen(p) - 6);
285.             break;
286.         case 'C'://Cookie
287.             if (strlen(p) > 8) {
288.                 char header[8];
289.                 ZeroMemory(header, sizeof(header));
290.                 memcpy(header, p, 6);
291.                 if (!strcmp(header, "Cookie")) {
292.                     memcpy(httpHeader->cookie, &p[8], strlen(p) - 8)
    ;
293.                 }
294.             }
295.             break;
```

```
296.         default:
297.             break;
298.         }
299.         p = strtok_s(NULL, delim, &ptr);
300.     }
301.     return true;
302. }
303. 
304. //***********************************
305. // Method:ConnectToServer
306. // FullName: ConnectToServer
307. // Access:public
308. // Returns:BOOL
309. // Qualifier: 根据主机创建目标服务器套接字，并连接
310. // Parameter: SOCKET * serverSocket
311. // Parameter: char * host
312. //***********************************
313. BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
314.     sockaddr_in serverAddr;
315.     serverAddr.sin_family = AF_INET;
316.     serverAddr.sin_port = htons(HTTP_PORT);
317.     HOSTENT* hostent = gethostbyname(host);
318.     if (!hostent) {
319.         return FALSE;
320.     }
321.     in_addr Inaddr = *((in_addr*)*hostent->h_addr_list);
322.     serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
323.     //printf("%s\n", inet_ntoa(Inaddr));
324.     *serverSocket = socket(AF_INET, SOCK_STREAM, 0);
325.     if (*serverSocket == INVALID_SOCKET) {
326.         return FALSE;
327.     }
328.     if (connect(*serverSocket, (SOCKADDR*)&serverAddr, sizeof(serve
    rAddr)) == SOCKET_ERROR) {
329.         closesocket(*serverSocket);
330.         return FALSE;
331.     }
332.     return TRUE;
333. }
334. 
335. //用 url 来为文件命名
336. void makeFilename(char* url, char* filename) {
337.     int count = 0;
338.     int len;
```

```
339.     while (*url != '\0') {
340.         if ((*url >= 'a' && *url <= 'z') || (*url >= 'A' && *url <=
     'Z') || (*url >= '0' && *url <= '9')) {
341.             *filename++ = *url;
342.             count++;
343.         }
344.         if (count >= 95)
345.             break;
346.         url++;
347.     }
348.     len = strlen(filename) + 1;
349.     strcat(filename,  ".txt");
350. }
351.
352. void getCache(char* buffer, char* filename) {
353.     char* p, * ptr, num[10], tempBuffer[MAXSIZE + 1];
354.     const char* delim = "\r\n";
355.     errno_t err;
356.     ZeroMemory(num, 10);
357.     ZeroMemory(tempBuffer, MAXSIZE + 1);
358.
359.     //对 tempbuffer 进行操作
360.     memcpy(tempBuffer, buffer, strlen(buffer));
361.
362.     p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
363.     memcpy(num, &p[9], 3);
364.     //printf("%s\n", buffer);
365.
366.     if (strcmp(num, "304") == 0) {  //主机返回的报文中的状态码为 304 时
     返回已缓存的内容
367.         ZeroMemory(buffer, strlen(buffer));
368.         printf("**************************************\n");
369.         printf("从本机获得缓存\n");
370.         FILE* in = NULL;
371.         err = fopen_s(&in, filename, "r");
372.         //从文件中获取缓存的内容
373.         if (in != NULL) {
374.             fread(buffer, sizeof(char), MAXSIZE, in);
375.             fclose(in);
376.         }
377.     }
378. }
379.
380. //创建 cache
```

```
381. void makeCache(char* buffer, char* url) {
382.     char* p, * ptr, num[10], tempBuffer[MAXSIZE + 1];
383.     const char* delim = "\r\n";
384.
385.     ZeroMemory(num, 10);
386.     ZeroMemory(tempBuffer, MAXSIZE + 1);
387.     memcpy(tempBuffer, buffer, strlen(buffer));
388.
389.     p = strtok_s(tempBuffer, delim, &ptr);//提取第一行
390.     memcpy(num, &p[9], 3);
391.     //printf("%s\n", num);
392.
393.     if (strcmp(num, "200") == 0) {  //状态码是 200 时缓存
394.         // 200 指成功访问，404 就是没成功
395.
396.         // 构建文件
397.         char filename[100];
398.         ZeroMemory(filename, 100);
399.         makeFilename(url, filename);
400.         printf("filename : %s\n", filename);
401.
402.         FILE *out, *fp;
403.         errno_t err;
404.         err = fopen_s(&fp, filename, "w");
405.         fwrite(buffer, sizeof(char), strlen(buffer), fp);
406.         fclose(fp);
407.         printf("************************************\n");
408.         printf("网页已经被缓存\n");
409.     }
410. }
411.
412. void makeNewHTTP(char* buffer, char* value) {
413.     const char* field = "Host";
414.     const char* newfield = "If-Modified-Since: ";
415.     //const char *delim = "\r\n";
416.     char temp[MAXSIZE];
417.     ZeroMemory(temp, MAXSIZE);
418.
419.     char* pos = strstr(buffer, field);
420.     int i = 0;
421.     for (i = 0; i < strlen(pos); i++) {
422.         temp[i] = pos[i];
423.     }
424.     *pos = '\0';
```

```
425.     while (*newfield != '\0') {  //插入 If-Modified-Since 字段
426.         *pos++ = *newfield++;
427.         //printf("%c", *pos);
428.     }
429.     while (*value != '\0') {
430.         *pos++ = *value++;
431.         //printf("%c", *pos);
432.     }
433.     *pos++ = '\r';
434.     *pos++ = '\n';
435.     for (i = 0; i < strlen(temp); i++) {
436.         *pos++ = temp[i];
437.     }
438.     //printf("报文首部变为\n%s\n", buffer);
439. }
440. 
441. //获取当前日期 date，存入 tempDate 里
442. void getDate(char* buffer, char* field, char* tempDate) {
443.     char* p, * ptr, temp[5];
444.     ZeroMemory(temp, 5);
445.     //*field = "If-Modified-Since";
446. 
447.     const char* delim = "\r\n";
448.     p = strtok_s(buffer, delim, &ptr); // 按行读取
449.     //printf("%s\n", p);
450.     int len = strlen(field) + 2;
451.     while (p) {
452.         if (strstr(p, field) != NULL) {
453.             // 如果 p 中包含 field 字串，将&p[6]copy 给 tempdate
454.             memcpy(tempDate, &p[len], strlen(p) - len);
455.             // printf("tempDate: %s\n", tempDate);
456.         }
457.         p = strtok_s(NULL, delim, &ptr);
458.     }
459. }
```