

**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 |
| 班级 |
| 任课教师 |
| 实验地点 |
| 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： |
| （注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）  熟悉并掌握 Socket 网络编程的过程与技术；深入理解 HTTP协议，掌握 HTTP 代理服务器的基本工作原理；掌握 HTTP 代理服务器设计与编程实现的基本技能。 |
| 实验内容： |
| 概述本次实验的主要内容，包含的实验项等。  (1) 设计并实现一个基本 HTTP 代理服务器。 要求在指定端口（例如8080） 接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址所指向的 HTTP 服务器（原服务器）， 接收 HTTP 服务器的响应报文，并将响应报文转发给对应的客户进行浏览。  (2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。 要求能缓存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since头行），向原服务器确认缓存对象是否是最新版本。  (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）。 |
| 实验过程： |
| 以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。   1. **Socket 编程的客户端和服务器端主要步骤**   **1.1 客户端主要步骤**  1. 根据目标服务器IP地址与端口号创建套接字，并连接服务器（三次握手）  2. 发送请求报文  3. 接收返回报文  4. 关闭连接  **1.2 服务器端主要步骤**  1.创建套接字，绑定套接字的本地IP地址和端口号，对端口进行监听。  2.从连接请求队列中取出一个连接请求，并同意连接。在TCP连接过程中进行了三次握手。  3.接收客户端请求  4.对请求进行响应，发送响应数据  5.关闭连接  **1.3 设置浏览器代理**  以IE浏览器设置为例：打开浏览器工具浏览器选项——连接——局域网设置——代理服务器。设置地址为127.0.0.1，端口号为10240。    **2.实现一个基本的HTTP代理服务器（基本原理、程序流程图和关键技术及解决方案）**  HTTP代理服务器用于一个网络终端（一般为客户端）通过代理服务与另一个网络终端（一般为服务器）进行非直接的连接。    **2.1 主要功能**  接收来自客户端的 HTTP 请求，并通过这个代理服务器将该请求转发给服务器；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端） 通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。  **2.2 基本原理**  代理服务器在指定端口（例如 8080）监听浏览器的访问请求（需要在客户端浏览器进行相应的设置），接收到浏览器对远程网站的浏览请求时，代理服务器开始在代理服务器的缓存中检索 URL 对应的对象（网页、图像等对象），找到对象文件后，提取该对象文件的最新被修改时间；代理服务器程序在客户的请求报文首部插入<If-Modified-Since: 对象文件的最新被修改时间>，并向原 Web 服务器转发修改后的请求报文。如果代理服务器没有该对象的缓存，则会直接向原服务器转发请求报文，并将原服务器返回的响应直接转发给客户端，同时将对象缓存到代理服务器中。代理服务器程序会根据缓存的时间、大小和提取记录等对缓存进行清理。  **2.3 程序流程图**    （以下函数直接借用参考代码，故不详细说明实现过程）   |  |  | | --- | --- | | 函数 | 功能 | | InitSocket()函数 | 初始化套接字 | | ProxyThread()函数 | 线程执行函数 | | ParseHttpHead()函数 | 解析TCP报文中的HTTP头部 | | ConnectToServer()函数 | 根据主机创建目标服务器套接字，并连接 |   **2.4 关键技术及解决方案**  1.关键技术：实现基本功能。  解决方案：见实验过程 2  2.关键技术：实现 cache 功能。  解决方案：见实验过程 3  3.关键技术：实现用户过滤，网站引导，网站屏蔽。  解决方案：见实验过程 4  **3.** **设计并实现一个支持Cache功能的HTTP代理服务器**   |  |  | | --- | --- | | 缓存相关参数 | 表示 | | haveCache | True是存在本地缓存，false表示不存在，用于判断是否是第一次请求 | | needCache | True是需要更新本地缓存，false表示不需要，用于判断是否需要更新缓存 |  |  |  | | --- | --- | | 缓存相关函数 | 功能 | | getfileDate()函数 | 访问本地文件，获取本地缓存中的日期 | | sendnewHTTP()函数 | 修改请求报文，添加 if-modified-since头行 | | checkfileCache()函数 | 检测主机返回的状态码，如果是304则从本地获取缓存进行转发，否则需要更新缓存 | | storefileCache()函数 | 检测主机返回的状态码，如果是200则本地获取缓存 |   **3.1 实现原理**  当客户第一次请求服务器资源，服务器成功返回资源，这时状态码为200。第一次收到服务器的响应对象，则将其缓存在本地文件中。如果不在有效期，客户端会请求服务器，重新获取。服务器会判断修改时间，如果没有修改过，就会返回状态码304，告诉客户端该资源仍然有效，客户端会直接使用缓存的资源。之后每一次收到服务器的响应对象，需要确认本地文件中的缓存是否是最新版本。如果HTTP中的状态码是304，则不需要更新本地缓存，直接将服务器的响应消息转发到客户端。如果HTTP中的状态码是200，则更新本地缓存。  **3.2 流程图**    **3.3 函数实现**  （1）getfileDate()函数  功能：访问本地文件，获取本地缓存中的日期  实现过程：由左图可知本地缓存报文格式，从文件对象in中读取文件内容至buffer中，匹配出”Date”所在行，即可获取本地缓存中的日期    （本地缓存报文格式） （实现代码）  （2）sendnewHTTP()函数  功能：修改请求报文，添加 if-modified-since头行  先查看请求报文格式：    实现过程：字符拼接处理，在请求报文中添加if-modified-since头行    （3）checkfileCache()函数  功能：检测主机返回的状态码，如果是304则从本地获取缓存进行转发，否则需要更新缓存  实现过程：首先使用strtok\_s函数获取状态码所在的行（第一行）    判断状态码是否为304,如果是，说明本地是最新缓存，读取本地缓存文件至buffer中，否则将needCache 置为FALSE，需要更新缓存。    （4）storefileCache()函数  功能：检测主机返回的状态码，如果是200则本地获取缓存  实现过程：首先使用strtok\_s函数获取状态码所在的行（第一行）    判断状态码是否为200,如果是，需要将报文缓存至本地    **4.实现扩展功能**  **4.1 网站过滤**  首先设置不允许访问网站的url    在处理客户端请求时，检查请求消息中的url是否被允许访问，如果不允许访问，则拒绝    **4.2 用户过滤**  将代理服务器的网络通信IP地址从INADDR\_ANY更改为特定的某个IP地址，从而只有该IP地址能通过  代理服务器访问外部网站，其他IP均不能    **4.3 网站引导**  首先设置目标网站和相应的钓鱼网站和主机名    在处理客户端请求时，将请求消息中的url和host替换成事先设置好的模拟网站的url和host |
| **验证过程以及实验结果**： |
| 采用演示截图、文字说明等方式，给出本次实验的实验结果。   1. 实现了一个基本 HTTP 代理服务器   运行代理服务器成功    以访问今日哈工大为例：     1. 实现一个支持 Cache 功能的 HTTP 代理服务器   例如：本科生服务网站的本地缓存文件：httpjweshiteducn110.txt       1. 实现了以下扩展功能 2. 网站过滤：允许/不允许访问某些网站；   以禁止访问今日哈工大为例       1. 用户过滤：支持/不支持某些用户访问外部网站；   例如只允许主机名为127.0.0.1访问外部网站    主机名为128.0.0.1访问外部网站失败    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）。  例如：将对统一身份认证网站引导至一个本科服务网站 |
| 问题讨论： |
| (实验过程中出现的问题，报错，异常)   1. 问题：不理解下列代码中为什么发送报文和接收报文的都是代理服务器，误以为代理服务器发送报文然后自己接收。     原因：因为程序是执行代理服务器的功能，send是向目标服务器发送报文，中间有一个目标服务器返回数据的过程（程序中未显示），等待目标服务器返回数据后，代理服务器接收目标服务器的报文，并非之前自身发送的报文。   1. 程序运行过程中突然异常终止，并报错如下：     原因：是在对文件读写操作发生错误，程序中根据主机请求的url来自动命名文件并缓存，经排查后发现是文件命名出错   1. 报错bind绑定套接字时一直绑定失败   原因：在网上查找解决方案，发现是套接字进程被占用了，程序运行窗口未关闭导致进程被占用。关闭窗口后不再出现报错。 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。 在本次实验中，我设计并实现一个基本 HTTP 代理服务器，实践了代理服务器在指定端口（例如8080）监听客户端的访问请求并将原服务器返回的响应直接转发给客户端的过程，更加深入了解客户端与服务器的通信机制。通过设计Cache功能，实时查看HTTP报文格式，我进一步了解了本地服务器的缓存机制原理与过程。通过实现网站过滤功能，用户过滤功能和钓鱼功能，我更加深入了解网络防火墙以及诈骗钓鱼网站等的工作原理，进一步提高了网络防诈骗的意识。 |

源代码

#include <stdio.h>

#include <Windows.h>

#include <process.h>

#include <string.h>

#include <tchar.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

//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));

};

};

//禁止访问的网站和钓鱼网站是否可以输入选择

char Invilid\_web[1024] = "http://today.hit.edu.cn/";//不允许访问的网站

//char Invilid\_web[1024] = "https://today.edu.cn/";

char Target\_web[1024] = "http://ids-hit-edu-cn.ivpn.hit.edu.cn";//统一身份认证网站

char Fish\_web[1024] = "http://jwes.hit.edu.cn/";//钓鱼网站

char Fish\_host[1024] = "jwes.hit.edu.cn"; //钓鱼主机名

BOOL InitSocket();

void ParseHttpHead(char\* buffer, HttpHeader\* httpHeader);

BOOL ConnectToServer(SOCKET\* serverSocket, char\* host);

unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter);

//代理相关参数

SOCKET ProxyServer;//代理服务器

sockaddr\_in ProxyServerAddr;//代理服务器地址

const int ProxyPort = 10240;//设置代理窗口

//缓存相关参数

boolean haveCache = false;

boolean needCache = true;

void getfileDate(FILE\* in, char\* tempDate);

void sendnewHTTP(char\* buffer, char\* datestring);

void makeFilename(char\* url, char\* filename);

void storefileCache(char\* buffer, char\* url);

void checkfileCache(char\* buffer, char\* filename);

//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;

};

//主程序

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;

//把socket设置成无效套接字

ProxyParam\* lpProxyParam;

HANDLE hThread;

DWORD dwThreadID;//unsigned long，无符号32位整型

//代理服务器不断监听

while (TRUE) {

acceptSocket = accept(ProxyServer, NULL, NULL);

lpProxyParam = new ProxyParam;//新的客户端和服务器端

if (lpProxyParam == NULL) {

continue;

}

lpProxyParam->clientSocket = acceptSocket;

//线程开始

hThread = (HANDLE)\_beginthreadex(NULL, 0,

&ProxyThread, (LPVOID)lpProxyParam, 0, 0);

CloseHandle(hThread);

Sleep(2000);

}

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;//地址族

ProxyServerAddr.sin\_port = htons(ProxyPort); // 设置代理端口

//ProxyServerAddr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;//设置IP地址

ProxyServerAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");//设置IP地址

//bind绑定

if (bind(ProxyServer, (SOCKADDR\*)& ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET\_ERROR) {

printf("绑定套接字失败\n");

return FALSE;

}

//listen监听，SOMAXCONN由系统来决定请求队列长度

if (listen(ProxyServer, SOMAXCONN) == SOCKET\_ERROR) {

printf("监听端口%d 失败", ProxyPort);

return FALSE;

}

return TRUE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: ProxyThread

// FullName: ProxyThread

// Access: public

// Returns: unsigned int \_\_stdcall

// Qualifier: 线程执行函数

// Parameter: LPVOID lpParameter

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter) {

char Buffer[MAXSIZE];

char\* CacheBuffer;

char\* DateBuffer;

char filename[100] = { 0 };

\_Post\_ \_Notnull\_ FILE\* in;

char date\_str[30]; //保存字段Date的值

ZeroMemory(Buffer, MAXSIZE);

SOCKADDR\_IN clientAddr;

int length = sizeof(SOCKADDR\_IN);

int recvSize;

int ret;

FILE\* fp;

//第一次接收客户端请求，将该请求缓存下来，存到本地文件中

recvSize = recv(((ProxyParam\*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);

HttpHeader\* httpHeader = new HttpHeader();

if (recvSize <= 0) {

goto error;

}

CacheBuffer = new char[recvSize + 1];

ZeroMemory(CacheBuffer, recvSize + 1);

memcpy(CacheBuffer, Buffer, recvSize);

ParseHttpHead(CacheBuffer, httpHeader);

//printf("HTTP请求报文如下：\n%s\n", Buffer);

ZeroMemory(date\_str, 30);

printf("httpHeader->url : %s\n", httpHeader->url);

makeFilename(httpHeader->url, filename);

//printf("filename是 %s\n", filename);

if ((fopen\_s(&in, filename, "r")) == 0) {

printf("\n有缓存\n");

//fread\_s(fileBuffer, MAXSIZE, sizeof(char), MAXSIZE, in);

getfileDate(in,date\_str);//得到本地缓存文件中的日期date\_str

fclose(in);

//printf("date\_str:%s\n", date\_str);

sendnewHTTP(Buffer, date\_str);

//向服务器发送一个请求，该请求需要增加 “If-Modified-Since” 字段

//服务器通过对比时间来判断缓存是否过期

haveCache = TRUE;

}

//printf("httpHeader的url是%s，不允许访问的是%s\n", httpHeader->url, Invilid\_web);

//网站过滤功能

if (strcmp(httpHeader->url, Invilid\_web) == 0) {

printf("%s网站被拒绝访问", Invilid\_web);

goto error;

}

//添加钓鱼功能

if (strstr(httpHeader->url, Target\_web) != NULL) {

printf("%s网站钓鱼成功，被转移至%s\n", Target\_web, Fish\_web);

memcpy(httpHeader->host, Fish\_host, strlen(Fish\_host) + 1);//替换主机名

memcpy(httpHeader->url, Fish\_web, strlen(Fish\_web) + 1);//替换url

}

//此时数据报存储在了httpHeader中

delete CacheBuffer;

//连接发送数据报所在的服务器

if (!ConnectToServer(&((ProxyParam\*)lpParameter)->serverSocket, httpHeader->host)) {

printf("连接目的服务器失败\n");

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;

}

//printf("服务器响应报文如下：\n%s\n", Buffer);

if (haveCache == true) {

checkfileCache(Buffer, httpHeader->url);

}

if (needCache == true) {

storefileCache(Buffer, httpHeader->url);

}

//将目标服务器返回的数据直接转发给客户端

ret = send(((ProxyParam\*)lpParameter)->clientSocket, Buffer, sizeof(Buffer), 0);

//错误处理

error:

printf("关闭套接字\n");

Sleep(200);

closesocket(((ProxyParam\*)lpParameter)->clientSocket);

closesocket(((ProxyParam\*)lpParameter)->serverSocket);

delete lpParameter;

\_endthreadex(0); //终止线程

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);

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));

\*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;

}

//访问本地文件，获取本地缓存中的日期

void getfileDate(FILE\* in, char\* tempDate) {

char field[5] = "Date";

char\* p, \* ptr, temp[5];

char buffer[MAXSIZE];

ZeroMemory(buffer, MAXSIZE);

fwrite(buffer, sizeof(char), MAXSIZE, in);

const char\* delim = "\r\n";//换行符

ZeroMemory(temp, 5);

p = strtok\_s(buffer, delim, &ptr);

int len = strlen(field) + 2;

while (p) {

if (strstr(p, field) != NULL) {//调用strstr后指针会指向匹配剩余的第一个字符

memcpy(tempDate, &p[len], strlen(p) - len);

return;

}

p = strtok\_s(NULL, delim, &ptr);

}

}

//改造HTTP请求报文

void sendnewHTTP(char\* buffer, char\* datestring) {

const char\* field = "Host";

const char\* newfield = "If-Modified-Since: ";

//const char \*delim = "\r\n";

char temp[MAXSIZE];

ZeroMemory(temp, MAXSIZE);

char\* pos = strstr(buffer, field);//获取请求报文段中Host后的部分信息

int i = 0;

for (i = 0; i < strlen(pos); i++) {

temp[i] = pos[i];//将pos复制给temp

}

\*pos = '\0';

while (\*newfield != '\0') { //插入If-Modified-Since字段

\*pos++ = \*newfield++;

}

while (\*datestring != '\0') {//插入对象文件的最新被修改时间

\*pos++ = \*datestring++;

}

\*pos++ = '\r';

\*pos++ = '\n';

for (i = 0; i < strlen(temp); i++) {

\*pos++ = temp[i];

}

}

//根据url构造文件名

void makeFilename(char\* url, char\* filename) {

while (\*url != '\0') {

if ('a' <= \*url && \*url <= 'z') {

\*filename++ = \*url;

}

url++;

}

strcat\_s(filename, strlen(filename) + 9, "110.txt");

}

//检测主机返回的状态码，如果是200则本地获取缓存

void storefileCache(char\* buffer, char\* url) {

char\* p, \* ptr, tempBuffer[MAXSIZE + 1];

//num中是状态码

const char\* delim = "\r\n";

ZeroMemory(tempBuffer, MAXSIZE + 1);

memcpy(tempBuffer, buffer, strlen(buffer));

p = strtok\_s(tempBuffer, delim, &ptr);//提取第一行

//printf("tempbuffer = %s\n", p);

if (strstr(tempBuffer, "200") != NULL) { //状态码是200时缓存

char filename[100] = { 0 };

makeFilename(url, filename);

printf("filename : %s\n", filename);

FILE\* out;

fopen\_s(&out, filename, "w+");

fwrite(buffer, sizeof(char), strlen(buffer), out);

fclose(out);

printf("\n===================网页已经被缓存==================\n\n");

}

}

//检测主机返回的状态码，如果是304则从本地获取缓存进行转发，否则需要更新缓存

void checkfileCache(char\* buffer, char\* filename) {

char\* p, \* ptr, tempBuffer[MAXSIZE + 1];

const char\* delim = "\r\n";

ZeroMemory(tempBuffer, MAXSIZE + 1);

memcpy(tempBuffer, buffer, strlen(buffer));

p = strtok\_s(tempBuffer, delim, &ptr);//提取状态码所在行

//主机返回的报文中的状态码为304时返回已缓存的内容

if (strstr(p, "304") != NULL) {

printf("\n=================从本机获得缓存====================\n\n");

ZeroMemory(buffer, strlen(buffer));

FILE\* in = NULL;

if ((fopen\_s(&in, filename, "r")) == 0) {

fread(buffer, sizeof(char), MAXSIZE, in);

fclose(in);

}

needCache = FALSE;

}

}