

**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 | 李国建 | | 院系 | 计算机学院 | | |
| 班级 | 1603107 | | 学号 | 1160300426 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | 格物207 | | 实验时间 | 2018/11/2 | | |
| 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。）  本次实验的主要目的。  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： |
| 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。（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 编程的客户端和服务器端主要步骤   客户端：  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  服务器端：  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. HTTP 代理服务器的基本原理   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。如图所示，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。  ds  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。   1. HTTP 代理服务器的程序流程图；      1. 实现 HTTP 代理服务器的关键技术及解决方案； 2. 关键技术：实现基本功能。   解决方案：  见下方实验过程（1）   1. 关键技术：实现 cache 功能。   解决方案：  见下方实验过程（2）   1. 关键技术：实现用户过滤，网站引导，网站屏蔽。   解决方案：  见下方实验过程（3）   1. HTTP 代理服务器实验验证过程以及实验结果；   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）实现一个代理服务器，在指定端口10240进行监听，接受来自客户端的 HTTP 请求，并返回来自服务器的响应。  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）实现一个支持 Cache 功能的 HTTP 代理服务器，利用外部文件存储来自服务器上的信息。  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。  （3）扩展 HTTP 代理服务器，支持如下功能：  （a）网站过滤：  对请求过来的 HTTP 报文头部进行检测，提取出其中的访问地址 url ，检测其是否为要被屏蔽的网址，如果是，则直接跳转到代码中的 erro 部分，即关闭套接字，断开此次连接。代码片段如下：  if (strcmp (httpHeader->url, INVILID\_WEBSITE) == 0) {  printf("\n=====================================\n\n");  printf("-------------Sorry!!!该网站已被屏蔽----------------\n");  goto error;  }  （b）用户过滤：  更改套接字绑定的主机地址，这样的话，只要不是从该地址访问代理服务器的客户端，都会被该代理服务器屏蔽，部分代码如下：  ProxyServerAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");//仅本机用户可访问服务器  （c）网站引导：  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）和 host 主机名来实现，部分代码如下：  if (strstr(httpHeader->url, FISHING\_WEB\_SRC) != NULL) {  printf("\n=====================================\n\n");  printf("-------------已从源网址：%s 转到 目的网址 ：%s ----------------\n", FISHING\_WEB\_SRC,FISHING\_WEB\_DEST);  memcpy(httpHeader->host, FISHING\_WEB\_HOST, strlen(FISHING\_WEB\_HOST) + 1);  memcpy(httpHeader->url, FISHING\_WEB\_DEST, strlen(FISHING\_WEB\_DEST));  } |
| 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. 基本 HTTP 代理服务器的实现，连接网址： “http://www.hit.edu.cn” ，显示结果：      1. 支持 Cache 功能的 HTTP 代理服务器实现，访问网址 <http://jwts.hit.edu.cn> 显示网址如下：     控制台信息如下：    缓存文件内容如下：     1. 扩展 HTTP 代理服务器： 2. 网站过滤，不允许访问网址 <http://www.qq.com/>，访问结果如图：     控制台信息如下：     1. 用户过滤，屏蔽本机地址，结果如图：     （c）网站引导（钓鱼）：将网址 <http://today.hit.edu.cn/> 的访问引导到网址 <http://jwts.hit.edu.cn/> ，运行结果如图：  C:\Users\13144\Documents\Tencent Files\2482354780\FileRecv\MobileFile\Image\6]@9KD6~J%7YOE{~I47@2HG.png  控制台输出信息： |
| 问题讨论： |
| 对实验过程中的思考问题进行讨论或回答。   1. 实验指导书的参考代码，#pragma comment(lib,"Ws2\_32.lib")无法使用，通过查询资料，得知：CodeBlocks 使用的是 MingGW 来编译，MingGW不支持#pragma comment(lib,"Ws2\_32.lib") 的写法，该命令是静态链接 Ws2\_32.lib 库， 可以在设置里，加上 -lws2\_32 或 -lwsock32。 2. CodeBlocks 无法使用 strtok\_s() 函数，这里需要切换成 strtok() 函数。 3. goto 语句后面，不能再新定义变量，否则，会出现一些问题。 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。  本次实验，让我对 socket 编程有了初步的了解，掌握了 HTTP 代理服务器的基本原理，对 HTTP 请求和响应原理有了更深的认识；同时，也对网站钓鱼、网站屏蔽等有了深刻的理解；更是感受到了 HTTP 缓存的神奇。让我对 socket 产生了浓厚的兴趣。 |

HTTP 代理服务器源代码（带有详细注释）

//#include "stdafx.h"

#include <stdio.h>

#include <Windows.h>

#include <process.h>

#include <string.h>

// Codeblocks 使用的是 MingGW 来编译，MingGW不支持#pragma comment(lib,"Ws2\_32.lib") 的写法

// 该命令是静态链接 Ws2\_32.lib 库， 可以在设置里，加上 -lws2\_32 或 -lwsock32

//#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 INVILID\_WEBSITE "http://www.qq.com/" //被屏蔽的网站

#define FISHING\_WEB\_SRC "http://today.hit.edu.cn/" //钓鱼的源网址

#define FISHING\_WEB\_DEST "http://jwts.hit.edu.cn/" //钓鱼的目的网址

#define FISHING\_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();

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

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

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

boolean ParseDate(char \*buffer, char \*field, char \*tempDate);

void makeNewHTTP(char \*buffer, char \*value);

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

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

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

//代理相关参数

SOCKET ProxyServer;

sockaddr\_in ProxyServerAddr;

const int ProxyPort = 10240;

//缓存相关参数

boolean haveCache = FALSE;

boolean needCache = TRUE;

char \* strArr[100];

//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;

};

int main(int argc, char\* 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;

ProxyParam \*lpProxyParam;

HANDLE hThread;

//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);

}

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; //WSADATA结构体中主要包含了系统所支持的Winsock版本信息

//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;

}

//LOBYTE()得到一个16bit数最低（最右边）那个字节

//HIBYTE()得到一个16bit数最高（最左边）那个字节

//判断打开的是否是2.2版本

if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) !=2) {

printf("不能找到正确的 winsock 版本\n");

WSACleanup();

return FALSE;

}

//AF\_INET,PF\_INET IPv4 Internet协议

//SOCK\_STREAM Tcp连接，提供序列化的、可靠的、双向连接的字节流。支持带外数据传输

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); //将整型变量从主机字节顺序转变成网络字节顺序

//屏蔽用户

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

ProxyServerAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");//仅本机用户可访问服务器

//ProxyServerAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.2"); //屏蔽用户

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: 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], fileBuffer[MAXSIZE];

char \*CacheBuffer;

HttpHeader\* httpHeader = new HttpHeader();

ZeroMemory(Buffer,MAXSIZE);

SOCKADDR\_IN clientAddr;

int length = sizeof(SOCKADDR\_IN);

int recvSize;

int ret;

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);

//解析http首部

ParseHttpHead(CacheBuffer, httpHeader);

//缓存

char \*DateBuffer;

DateBuffer = (char\*)malloc(MAXSIZE);

ZeroMemory(DateBuffer, strlen(Buffer) + 1);

memcpy(DateBuffer, Buffer, strlen(Buffer) + 1);

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

char filename[100];

ZeroMemory(filename, 100);

makeFilename(httpHeader->url, filename);

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

char \*field = "Date";

char date\_str[30]; //保存字段Date的值

ZeroMemory(date\_str, 30);

ZeroMemory(fileBuffer, MAXSIZE);

FILE \*in;

if ((in = fopen(filename, "rb")) != NULL) {

printf("\n有缓存\n");

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

fclose(in);

ParseDate(fileBuffer, field, date\_str);

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

makeNewHTTP(Buffer, date\_str);

haveCache = TRUE;

goto success;

}

//网站过滤：屏蔽一个网站

if (strcmp (httpHeader->url, INVILID\_WEBSITE) == 0) {

printf("\n=====================================\n\n");

printf("-------------Sorry!!!该网站已被屏蔽----------------\n");

goto error;

}

//网站引导：将访问网址转到其他网站

if (strstr(httpHeader->url, FISHING\_WEB\_SRC) != NULL) {

printf("\n=====================================\n\n");

printf("-------------已从源网址：%s 转到 目的网址 ：%s ----------------\n", FISHING\_WEB\_SRC,FISHING\_WEB\_DEST);

// 至于为什么要加一，我也不知道，只能说设成不加一不好使 只要大于他原来的长度就可以

memcpy(httpHeader->host, FISHING\_WEB\_HOST, strlen(FISHING\_WEB\_HOST) + 1);

memcpy(httpHeader->url, FISHING\_WEB\_DEST, strlen(FISHING\_WEB\_DEST));

//memcpy(httpHeader->host, "jwts.hit.edu.cn", strlen("jwts.hit.edu.cn")); //这行不可以用，不知道为什么。。。

}

delete CacheBuffer;

delete DateBuffer;

success:

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

printf("连接目标服务器失败！！！\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){

printf("返回目标服务器的数据失败！！！\n");

goto error;

}

//有缓存时，判断返回的状态码是否是304，若是则将缓存的内容发送给客户端

if (haveCache == TRUE) {

getCache(Buffer, filename);

}

if (needCache == TRUE) {

makeCache(Buffer, httpHeader->url); //缓存报文

}

//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");

delete Buffer;

delete fileBuffer;

delete filename;

Sleep(200);

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

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

delete lpParameter;

\_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;

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

p = strtok(buffer,delim); // 第一次调用，第一个参数为被分解的字符串

//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); //'Get' 和 'HTTP/1.1' 各占 3 和 8 个，再加上俩空格，一共13个

}

else if(p[0] == 'P'){

//POST 方式

memcpy(httpHeader->method,"POST",4);

memcpy(httpHeader->url,&p[5],strlen(p) - 14); //'Post' 和 'HTTP/1.1' 各占 4 和 8 个，再加上俩空格，一共14个

}

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

p = strtok(NULL,delim); // 第二次调用，需要将第一个参数设为 NULL

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(NULL,delim);

}

}

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

// 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;

}

//分析HTTP头部的field字段，如果包含该field则返回true，并获取日期

boolean ParseDate(char \*buffer, char \*field, char \*tempDate) {

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

//\*field = "If-Modified-Since";

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

ZeroMemory(temp, 5);

p = strtok(buffer, delim);

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

int len = strlen(field) + 2;

while (p) {

if (strstr(p, field) != NULL) {

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

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

return TRUE;

}

p = strtok(NULL, delim);

}

return TRUE;

}

//改造HTTP请求报文

void makeNewHTTP(char \*buffer, char \*value) {

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);

int i = 0;

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

temp[i] = pos[i];

}

\*pos = '\0';

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

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

}

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

\*pos++ = \*value++;

}

\*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 (\*url != '/' && \*url != ':' && \*url != '.') {

\*filename++ = \*url;

}

url++;

}

strcat(filename, ".txt");

}

//进行缓存

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

char \*p, \*ptr, num[10], tempBuffer[MAXSIZE + 1];

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

ZeroMemory(num, 10);

ZeroMemory(tempBuffer, MAXSIZE + 1);

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

p = strtok(tempBuffer, delim);//提取第一行

memcpy(num, &p[9], 3);

if (strcmp(num, "200") == 0) { //状态码是200时缓存

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

char filename[100] = { 0 }; // 构造文件名

makeFilename(url, filename);

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

FILE \*out;

out = fopen(filename, "w");

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

fclose(out);

printf("\n=====================================\n\n");

printf("\n网页已经被缓存\n");

}

}

//获取缓存

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

char \*p, \*ptr, num[10], tempBuffer[MAXSIZE + 1];

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

ZeroMemory(num, 10);

ZeroMemory(tempBuffer, MAXSIZE + 1);

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

p = strtok(tempBuffer, delim);//提取第一行

memcpy(num, &p[9], 3);

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

printf("\n=====================================\n\n");

printf("从本机获得缓存\n");

ZeroMemory(buffer, strlen(buffer));

FILE \*in = NULL;

if ((in = fopen(filename, "r")) != NULL) {

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

fclose(in);

}

needCache = FALSE;

}

}