

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

****

|  |
| --- |
| 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) 4. 网站过滤: 允许/不允许访问某些网站; 5. 用户过滤: 支持/不支持某些用户访问外部网站: 6. 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：**     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。   1. **初始化socket：** 2. 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。     1. 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；     1. **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。    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。    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。     1. **解析http头部：**   主要通过ParseHttpHead函数来实现。    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。    接着循环提取报文，提取http首部的其他必要信息。   1. **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。    **选做部分：**   1. **支持cache功能：**   首先我们使用makefliename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。    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。    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。    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。     1. **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。       1. **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。       1. **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。 |
| 实验结果： |
| 1. **实现代理服务器功能，访问计算学部官网**        1. **实现缓存，并更新缓存文件**   缓存文件如下图所示：    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：     1. **屏蔽网站今日哈工大**   可以看到提示，该网站已被屏蔽。       1. **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：     1. **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** |
| 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 7. 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  1. 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  1. 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。   1. HTTP代理服务器流程图      1. 实现HTTP代理服务器的关键技术及解决方案 2. 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。   1. 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>
4. #include <string.h>
6. #pragma comment(lib,"Ws2\_32.lib")
7. #define MAXSIZE 65507 //发送数据报⽂的最⼤⻓度
8. #define HTTP\_PORT 80 //http 服务器端⼝
10. #define invalid\_website "http://today.hit.edu.cn/" //屏蔽网址
11. #define fish\_web\_src "http://www.7k7k.com/" //钓鱼源网址
12. #define fish\_web\_url "http://jwts.hit.edu.cn/" //钓鱼目的网址
13. #define fish\_web\_host "jwts.hit.edu.cn" //钓鱼目的地址的主机名
15. //Http 重要头部数据
16. **struct** HttpHeader {
17. **char** method[4]; // POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
18. **char** url[1024]; // 请求的 url
19. **char** host[1024]; // ⽬标主机
20. **char** cookie[1024 \* 10]; //cookie
21. HttpHeader() {
22. ZeroMemory(**this**, **sizeof**(HttpHeader));
23. }
24. };
26. **BOOL** InitSocket();
27. **int** ParseHttpHead(**char**\* buffer, HttpHeader\* httpHeader);
28. **BOOL** ConnectToServer(SOCKET\* serverSocket, **char**\* host);
29. unsigned **int** \_\_stdcall ProxyThread(**LPVOID** lpParameter);
30. **void** makeFilename(**char**\* url, **char**\* filename);
31. **void** getCache(**char**\* buffer, **char**\* filename);
32. **void** makeCache(**char**\* buffer, **char**\* url);
33. **void** makeNewHTTP(**char**\* buffer, **char**\* value);
34. **void** getDate(**char**\* buffer, **char**\* field, **char**\* tempDate);
36. //代理相关参数
37. SOCKET ProxyServer;
38. sockaddr\_in ProxyServerAddr;
39. **const** **int** ProxyPort = 10240;
40. **const** **char**\* forbid\_user[10] = { "127.0.0.1" };  //被屏蔽的用户IP
41. **bool** flag = **true**;  //用户过滤开启或关闭
43. //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
44. //可以使⽤线程池技术提⾼服务器效率
45. //const int ProxyThreadMaxNum = 20;
46. //HANDLE ProxyThreadHandle[ProxyThreadMaxNum] = {0};
47. //DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};
49. **struct** ProxyParam {
50. SOCKET clientSocket;
51. SOCKET serverSocket;
52. };
54. **int** main(**int** argc, **char**\* argv[])
55. {
56. sockaddr\_in addr\_in;
57. **int** addr\_len = **sizeof**(SOCKADDR);
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, &(addr\_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, (**LPVOID**)lpProxyParam, 0, 0);
86. CloseHandle(hThread);
87. Sleep(200);
88. }
89. closesocket(ProxyServer);
90. WSACleanup();
91. **return** 0;
92. }
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. }
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);
154. SOCKADDR\_IN clientAddr;
155. **int** length = **sizeof**(SOCKADDR\_IN);
157. **char** fileBuffer[MAXSIZE];
158. **char** \*filename = (**char**\*)calloc(100, **sizeof**(**char**));
159. **FILE**\* in;
161. **char**\* field = (**char**\*)"Date";
162. **char** date\_str[30];
163. ZeroMemory(date\_str, 30);
165. **BOOL** haveCache = **false**;
166. HttpHeader\* httpHeader = **new** HttpHeader();
168. **int** recvSize;
169. **int** ret;
171. //从客户端获得http报文，存入Buffer
172. recvSize = recv(((ProxyParam\*)lpParameter)->clientSocket, Buffer, MAXSIZE, 0);
173. **if** (recvSize <= 0) {
174. **goto** error;
175. }
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;
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. }
201. **if** (strcmp(httpHeader->url, invalid\_website) == 0)
202. {
203. printf("\*\*\*\*\*\*\*\*\*\*\*该网站已被屏蔽\*\*\*\*\*\*\*\*\*\*\*\n");
204. **goto** error;
205. }
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\_host) + 1);
211. memcpy(httpHeader->url, fish\_web\_url, strlen(fish\_web\_url));
212. }
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);
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, strlen(Buffer) + 1, 0);
221. //等待⽬标服务器返回数据
222. recvSize = recv(((ProxyParam\*)lpParameter)->serverSocket, Buffer, MAXSIZE, 0);
223. **if** (recvSize <= 0) {
224. **goto** error;
225. }
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. }
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, **sizeof**(Buffer), 0);
238. printf("数据已经发送给用户\n\n");
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. }
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";
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. }
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**(serverAddr)) == SOCKET\_ERROR) {
329. closesocket(\*serverSocket);
330. **return** FALSE;
331. }
332. **return** TRUE;
333. }
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. }
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);
359. //对tempbuffer进行操作
360. memcpy(tempBuffer, buffer, strlen(buffer));
362. p = strtok\_s(tempBuffer, delim, &ptr);//提取第一行
363. memcpy(num, &p[9], 3);
364. //printf("%s\n", buffer);
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. }
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";
385. ZeroMemory(num, 10);
386. ZeroMemory(tempBuffer, MAXSIZE + 1);
387. memcpy(tempBuffer, buffer, strlen(buffer));
389. p = strtok\_s(tempBuffer, delim, &ptr);//提取第一行
390. memcpy(num, &p[9], 3);
391. //printf("%s\n", num);
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
396. // 构建文件
397. **char** filename[100];
398. ZeroMemory(filename, 100);
399. makeFilename(url, filename);
400. printf("filename : %s\n", filename);
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. }
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);
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. }
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";
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. }