|  |
| --- |
| 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10240）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： |
| **一、设置浏览器使用代理**   1. 打开设置→网络和Internet→代理→手动设置代理→使用代理服务器→设置 2. 设置代理 IP 地址为 127.0.0.1，端口号为 10240，保存     **二、HTTP 代理服务器基本功能实现**  启动和初始化代理服务器，调用 InitSocket() 函数初始化网络套接字，如果 InitSocket 成功，代理服务器开始监听指定端口的消息。接着，程序创建套接字和线程，不断监听并使用 Winsock 中的 accept() 函数来接受客户端的连接请求，并创建一个与客户端连接的新的 Socket。  若接收到连接请求，记录客户端的 IP 地址，为其创建一个新的 ProxyParam 结构体，其中包含客户端套接字和服务器套接字（初始为 INVALID\_SOCKET）。然后，程序创建一个新的线程，处理客户端的HTTP请求，与目标服务器通信，并将响应返回给客户端。  线程执行函数 ProxyThread() 用于执行线程的实际工作。线程处理完成后，调用 CloseHandle() 函数关闭客户端和服务器的套接字，并清理分配的内存，主线程会继续运行，不断接受新的连接请求，并为每个请求创建新线程处理。  主要调用函数如下：  （1）InitSocket：初始化 Winsock 库，创建一个套接字，并将其绑定到指定的端口上，然后监听来自客户端的连接请求。  （2）ProxyThread：执行线程的实际工作，接收客户端发送的请求和数据，解析HTTP头部，检查缓存，连接目标服务器，转发请求和响应等。  （3）ParseHttpHead：解析 TCP 报文中的的 HTTP 请求头部，提取出请求方法（GET / POST）、url、主机名和 Cookie 等信息。  （4）ConnectToServer：根据主机创建一个目标服务器套接字，并尝试连接。  **三、HTTP 代理服务器 Cache 功能实现**  线程执行函数 ProxyThread() 接收来自客户端的 HTTP 请求，解析报文头部，并存储在 HttpHeader 结构体中。调用 makeFilename() 函数，根据请求的 url 构造一个文件名，并尝试打开该文件。如果文件存在，则说明本地已有缓存，程序会通过 getfileDate() 函数读取缓存文件中的日期信息，并使用 sendnewHTTP() 函数构造一个新的 HTTP 请求，添加 If-Modified-Since 头部（用于向服务器确认缓存对象是否为最新版本），之后将该请求转发至目标服务器；如果本地没有缓存，则直接转发原始请求到目标服务器。  接收目标服务器的响应，若状态码为304，说明资源自上次缓存以来没有变化，可以直接使用本地缓存；若状态码为200，说明资源已被修改（缓存过期），需要更新本地缓存。最后将响应转发回客户端。  **四、HTTP 代理服务器扩展功能实现**  **1. 网站过滤（允许/不允许访问某些网站）**  在 ProxyThread() 函数中，程序解析 HTTP 报文头部后，调用 inWebShield() 函数检查用户请求的 url 是否在shieldWeb 数组（网站黑名单）中。    若该 url 在黑名单中，代理服务器将拒绝连接，并向客户端发送一个错误消息，然后终止该线程；若该 url 不在黑名单中，则正常执行。  **2. 用户过滤（支持/不支持某些用户访问外部网站）**  代理服务器不断监听指定端口的信息，若收到客户端的连接请求，会获取客户端的 IP地址，并调用 inUserShield() 函数检查该 IP 地址是否在 shieldIp 数组（用户黑名单）中。    若客户端 IP 在黑名单中，则屏蔽该请求，显示拒绝访问，并继续监听下一个连接请求；若客户端 IP 不在黑名单中，则处理和转发该 HTTP 请求和响应。  **3. 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））**  在 ProxyThread() 函数中，如果请求的 url 包含预先设置的目标网站（Target\_web），代理服务器将修改 HTTP 请求头部，将请求重定向到钓鱼网站（Fish\_web）。    这一操作通过修改 HttpHeader 结构体中的 host 和 url 字段实现，从而改变客户端实际访问的目标地址。  **五、HTTP 代理服务器程序流程图**    **程序源码：**   1. #include <stdio.h> 2. #include <tchar.h> 3. #include <Windows.h> 4. #include <process.h> 5. #pragma comment(lib, "Ws2\_32.lib") 6. #define MAXSIZE 65507 *// 发送数据报文的最大长度* 7. #define HTTP\_PORT 80 *// http 服务器端口* 8. *// 代理相关参数* 9. const int proxyPort = 10240; 10. SOCKET ProxyServer; *// 代理服务器* 11. int addrLen = 32; 12. sockaddr\_in ProxyServerAddr; *// 代理服务器地址，sockaddr\_in 结构体用于存储 IPv4 地址信息* 13. *// 缓存相关参数* 14. boolean haveCache = false; *// 标记是否有缓存* 15. boolean needCache = true; *// 标记是否需要缓存* 16. struct ProxyParam 17. { 18. SOCKET clientSocket; 19. SOCKET serverSocket; 20. }; 21. *// http重要头部数据* 22. struct HttpHeader { 23. char method[4];  *// POST 或 GET* 24. char url[1024]; 25. char host[1024]; *// 目标主机* 26. char cookie[1024 \* 10]; 27. HttpHeader() { 28. ZeroMemory(this, sizeof(HttpHeader)); 29. }; 30. }; 31. BOOL InitSocket(); 32. unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter); 33. void ParseHttpHead(char\* buffer, HttpHeader\* httpHeader); 34. void makeFilename(char\* url, char\* filename); 35. void getfileDate(FILE\* in, char\* tempDate); 36. void sendnewHTTP(char\* buffer, char\* datestring); 37. BOOL ConnectToServer(SOCKET\* serverSocket, char\* host); 38. void checkfileCache(char\* buffer, char\* filename); 39. void storefileCache(char\* buffer, char\* url); 40. int inWebShield(char\* ip); 41. int inUserShield(char\* ip); 42. *// 用户过滤（黑名单）* 43. char shieldIp[][20] = { 44. *//{"127.0.0.1"},* 45. { "123.4.5.6" } 46. }; 47. *// 网站过滤（黑名单）* 48. char shieldWeb[][1024] = { 49. {"http://jwes.hit.edu.cn/"} 50. }; 51. *// 网站引导（钓鱼）* 52. char Target\_web[1024] = "http://www.baidu.com";  *// 目标网站* 53. char Fish\_web[1024] = "http://szsj.hit.edu.cn/"; *// 钓鱼网站* 54. char Fish\_host[1024] = "szsj.hit.edu.cn";   *// 钓鱼主机名* 55. int \_tmain(int argc, \_TCHAR\* argv[]) { 56. printf("代理服务器启动\n"); 57. printf("初始化……\n"); 58. if (!InitSocket()) { 59. printf("错误！！！：socket 初始`化失败！\n"); 60. return -1; 61. } 62. printf("代理服务器正在运行，监听端口：%d\n", proxyPort); 63. *// 设置SOCKET为无效套接字* 64. SOCKET acceptSocket = INVALID\_SOCKET; 65. *// 获取用户IP* 66. SOCKADDR\_IN acceptAddr; 67. ProxyParam\* lpProxyParam; 68. HANDLE hThread; 69. DWORD dwThreadID; 70. *// 代理服务器保持监听* 71. while (TRUE) { 72. *// 使用 Winsock 中的 accept() 函数来接受客户端的连接请求，并创建一个与客户端连接的新的套接字* 73. *// 在调用 accept() 函数之前，必须设置 addrlen 的初始值为 sizeof(acceptAddr)，* 74. *// 以确保函数能够正确地获取客户端地址的信息。* 75. addrLen = sizeof(acceptAddr); 76. acceptSocket = accept(ProxyServer, (SOCKADDR\*)&acceptAddr, &addrLen); 77. *// inet\_ntoa()* 78. *// 将一个 IPv4 的网络地址结构体（struct in\_addr）中的 IP 地址转换成一个以点分十进制表示的字符串形式* 79. printf("\n---------------------------------------------\n            用户IP地址为 %s \n---------------------------------------------\n", inet\_ntoa(acceptAddr.sin\_addr)); 80. *// 屏蔽用户IP* 81. if (inUserShield(inet\_ntoa(acceptAddr.sin\_addr)) != -1) { 82. printf("\n\n-----------拒绝访问：该用户无访问权限！------------\n\n"); 83. } 84. else { 85. *// 创建一个新的线程* 86. *// 执行此行代码后，应记得及时管理已分配的内存，以防止内存泄漏。* 87. *// 通常情况下，这涉及在不再需要时释放内存，使用 delete 关键字* 88. lpProxyParam = new ProxyParam; 89. if (lpProxyParam == NULL) { 90. continue; 91. } 92. lpProxyParam->clientSocket = acceptSocket; 93. *// 线程开始* 94. *// &ProxyThread：用于执行线程的实际工作。* 95. *// (LPVOID)lpProxyParam：这是传递给线程函数的参数。* 96. hThread = (HANDLE)\_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, 0, 0); 97. CloseHandle(hThread); 98. } 99. Sleep(10000); 100. } 101. closesocket(ProxyServer); 102. WSACleanup(); 103. return 0; 104. } 105. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 106. *// Method: InitSocket* 107. *// FullName: InitSocket* 108. *// Access: public* 109. *// Returns: BOOL* 110. *// Qualifier: 初始化套接字* 111. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 112. BOOL InitSocket() { 113. *// 创建套接字库（必须）* 114. *// 在 Winsock 编程中，wVersionRequested 通常用于指定应用程序需要使用的 Winsock 版本。(2.2)* 115. *// WORD 是 Windows 数据类型之一，表示一个16位无符号整数。* 116. WORD wVersionRequested = MAKEWORD(2, 2); 117. *// 用于存储 Winsock 库的相关信息，包括库的版本号等。* 118. *// 以便在函数成功初始化 Winsock 库后，将相关信息填充到 wsaData 结构中* 119. WSAData wsaData; 120. *// 套接字加载时的错误提示* 121. *// 加载dll文件Socket库* 122. int err = WSAStartup(wVersionRequested, &wsaData); 123. if (err != 0) 124. { 125. *// 找不到winsock.dll* 126. printf("错误：加载 winsock 失败，错误代码： %d\n", WSAGetLastError()); 127. return FALSE; 128. } 129. if (wsaData.wVersion != wVersionRequested) { 130. printf("错误：不能找到正确的 winsock 版本\n"); 131. WSACleanup(); 132. return FALSE; 133. } 134. *// 创建套接字* 135. *// AF\_INET：地址族（Address Family），指定了套接字的地址类型，这里是 IPv4 地址族。* 136. *// SOCK\_STREAM：套接字类型，这里指定了使用 TCP 协议进行可靠的面向连接的通信。* 137. *// 0：指定套接字的协议。在大多数情况下，都可以设置为 0，表示根据前两个参数自动选择合适的协议。* 138. ProxyServer = socket(AF\_INET, SOCK\_STREAM, 0); 139. if (ProxyServer == INVALID\_SOCKET) { 140. printf("错误：创建套接字失败，错误代码为 %d\n", WSAGetLastError()); 141. return FALSE; 142. } 143. ProxyServerAddr.sin\_family = AF\_INET; 144. *// htons() 函数用于将主机字节序（host byte order）转换为网络字节序（network byte order）* 145. *// 确保端口号以正确的字节序存储。* 146. ProxyServerAddr.sin\_port = htons(proxyPort);*// 设置代理窗口* 147. ProxyServerAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");*// 设置IP地址* 148. *// bind 绑定* 149. if (bind(ProxyServer, (SOCKADDR\*)&ProxyServerAddr, sizeof(SOCKADDR)) == SOCKET\_ERROR) { 150. printf("错误：绑定套接字失败\n"); 151. return FALSE; 152. } 153. *// listen 监听，SOMAXCONN由系统来决定请求队列长度* 154. *// SOMAXCONN 是一个系统常量，表示系统能够处理的最大连接数* 155. if (listen(ProxyServer, SOMAXCONN) == SOCKET\_ERROR) { 156. printf("错误：监听端口 %d 失败\n", proxyPort); 157. return FALSE; 158. } 159. return TRUE; 160. } 161. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 162. *// Method: ProxyThread* 163. *// FullName: ProxyThread* 164. *// Access: public* 165. *// Returns: unsigned int \_\_stdcall* 166. *// Qualifier: 线程执行函数* 167. *// Parameter: LPVOID lpParameter* 168. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 169. *// 用于执行线程的实际工作* 170. *// 在使用 \_\_stdcall 调用约定时，参数是从右向左依次压入堆栈，函数返回后由调用者负责清理堆栈。* 171. unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter) { 172. char buffer[MAXSIZE]; *// 一个缓冲区，用于存储接收到的数据。* 173. ZeroMemory(buffer, MAXSIZE); 174. char\* cacheBuffer; 175. char\* dateBuffer; 176. char filename[100] = { 0 }; 177. \_Post\_ \_Notnull\_ FILE\* in; 178. char date\_str[30]; *// 保存字段Date的值* 179. SOCKADDR\_IN clientAddr; 180. int lenth = sizeof(SOCKADDR\_IN); 181. int recvSize; *// 成功接受的字节数* 182. int ret;  *// 成功发送的字节数* 183. FILE\* fp; 184. *// 第一次接收客户端请求，将该请求缓存到本地文件中* 185. recvSize = recv(((ProxyParam\*)lpParameter)->clientSocket, buffer, MAXSIZE, 0); 186. HttpHeader\* httpHeader = new HttpHeader(); 187. if (recvSize <= 0) { 188. goto error; 189. } 190. cacheBuffer = new char[recvSize + 1]; 191. ZeroMemory(cacheBuffer, recvSize + 1); 192. *// 将从 buffer 中接收到的数据复制到 cacheBuffer 中* 193. memcpy(cacheBuffer, buffer, recvSize); 194. ParseHttpHead(cacheBuffer, httpHeader); *// 解析HTTP报文头部* 195. ZeroMemory(date\_str, 30); 196. printf("用户请求url : %s\n", httpHeader->url); 197. makeFilename(httpHeader->url, filename); 198. *// 成功打开文件* 199. if ((fopen\_s(&in, filename, "r")) == 0) { 200. printf("\n命中，本地已有缓存文件\n"); 201. getfileDate(in, date\_str); *// 得到本地缓存文件中的日期date\_str* 202. fclose(in); 203. *// 向服务器发送一个请求，该请求需要增加 “If-Modified-Since” 字段* 204. *// 服务器通过对比时间来判断缓存是否过期* 205. sendnewHTTP(buffer, date\_str); 206. haveCache = TRUE; 207. } 208. *// 没有缓存文件* 209. else { 210. printf("\n未命中，本地没有缓存文件\n"); 211. } 212. *// 网站黑名单* 213. if (inWebShield(httpHeader->url) != -1) { 214. printf("%s 该网站已被屏蔽！\n", shieldWeb[inWebShield(httpHeader->url)]); 215. goto error; 216. } 217. *// 钓鱼网站* 218. if (strstr(httpHeader->url, Target\_web) != NULL) { 219. printf("\n\n网站钓鱼成功！\n%s被转移至%s\n\n", Target\_web, Fish\_web); 220. memcpy(httpHeader->host, Fish\_host, strlen(Fish\_host) + 1); *// 替换主机名* 221. memcpy(httpHeader->url, Fish\_web, strlen(Fish\_web) + 1); *// 替换url* 222. } 223. *// 此时数据报存储在了httpHeader中* 224. delete cacheBuffer; 225. *// 连接发送数据报所在的服务器* 226. if (!ConnectToServer(&((ProxyParam\*)lpParameter)->serverSocket, httpHeader->host)) { 227. printf("错误：连接目的服务器失败\n"); 228. goto error; 229. } 230. printf("代理成功连接主机： %s \n", httpHeader->host); 231. *//将客户端发送的 HTTP 数据报文转发给目标服务器* 232. ret = send(((ProxyParam\*)lpParameter)->serverSocket, buffer, strlen(buffer) + 1, 0); 233. *//等待目标服务器返回数据* 234. recvSize = recv(((ProxyParam\*)lpParameter)->serverSocket, buffer, MAXSIZE, 0); 235. if (recvSize <= 0) { 236. goto error; 237. } 238. *// 如果已有缓存* 239. if (haveCache == true) { 240. checkfileCache(buffer, httpHeader->url); 241. } 242. *// 如果需要缓存* 243. if (needCache == true) { 244. storefileCache(buffer, httpHeader->url); 245. } 246. *// 将目标服务器返回的数据直接转发给客户端* 247. ret = send(((ProxyParam\*)lpParameter)->clientSocket, buffer, sizeof(buffer), 0); 248. *// 错误处理* 249. error: 250. printf("套接字已关闭\n\n"); 251. Sleep(200); 252. closesocket(((ProxyParam\*)lpParameter)->clientSocket); 253. closesocket(((ProxyParam\*)lpParameter)->serverSocket); 254. delete lpParameter; 255. \_endthreadex(0); *//终止线程* 256. return 0; 257. } 258. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 259. *// Method: ParseHttpHead* 260. *// FullName: ParseHttpHead* 261. *// Access: public* 262. *// Returns: void* 263. *// Qualifier: 解析 TCP 报文中的 HTTP 头部* 264. *// Parameter: char \* buffer* 265. *// Parameter: HttpHeader \* httpHeader* 266. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 267. void ParseHttpHead(char\* buffer, HttpHeader\* httpHeader) { 268. char\* p; 269. char\* ptr; 270. const char\* delim = "\r\n"; 271. *// 提取第一行* 272. p = strtok\_s(buffer, delim, &ptr); 273. printf("%s\n", p); 274. *// GET 方式* 275. if (p[0] == 'G') { 276. memcpy(httpHeader->method, "GET", 3); 277. memcpy(httpHeader->url, &p[4], strlen(p) - 13); 278. } 279. *// POST 方式* 280. else if (p[0] == 'P') { 281. memcpy(httpHeader->method, "POST", 4); 282. memcpy(httpHeader->url, &p[5], strlen(p) - 14); 283. } 284. printf("%s\n", httpHeader->url); 285. p = strtok\_s(NULL, delim, &ptr); 286. while (p) { 287. switch (p[0]) { 288. *//Host* 289. case 'H': 290. memcpy(httpHeader->host, &p[6], strlen(p) - 6); 291. break; 292. *//Cookie* 293. case 'C': 294. if (strlen(p) > 8) { 295. char header[8]; 296. ZeroMemory(header, sizeof(header)); 297. memcpy(header, p, 6); 298. if (!strcmp(header, "Cookie")) { 299. memcpy(httpHeader->cookie, &p[8], strlen(p) - 8); 300. } 301. } 302. break; 303. default: 304. break; 305. } 306. p = strtok\_s(NULL, delim, &ptr); 307. } 308. } 309. *// 根据url构造文件名* 310. void makeFilename(char\* url, char\* filename) { 311. while (\*url != '\0') { 312. if ('a' <= \*url && \*url <= 'z') { 313. \*filename++ = \*url; 314. } 315. url++; 316. } 317. strcat\_s(filename, strlen(filename) + 9, ".txt"); 318. } 319. *// 访问本地文件，获取本地缓存中的日期* 320. void getfileDate(FILE\* in, char\* tempDate) { 321. char field[5] = "Date"; 322. char\* p, \* ptr, temp[5]; 323. char buffer[MAXSIZE]; 324. ZeroMemory(buffer, MAXSIZE); 325. fread(buffer, sizeof(char), MAXSIZE, in); 326. const char\* delim = "\r\n"; *// 换行符* 327. ZeroMemory(temp, 5); 328. p = strtok\_s(buffer, delim, &ptr); 329. int len = strlen(field) + 2; 330. while (p) { 331. if (strstr(p, field) != NULL) { *// 调用strstr后指针会指向匹配剩余的第一个字符* 332. memcpy(tempDate, &p[len], strlen(p) - len); 333. return; 334. } 335. p = strtok\_s(NULL, delim, &ptr); 336. } 337. } 338. *// 修改HTTP请求报文* 339. void sendnewHTTP(char\* buffer, char\* datestring) { 340. const char\* field = "Host"; 341. const char\* newfield = "If-Modified-Since: "; 342. char temp[MAXSIZE]; 343. ZeroMemory(temp, MAXSIZE); 344. *// 获取请求报文段中Host后的部分信息* 345. char\* pos = strstr(buffer, field); 346. *// 将pos复制给temp* 347. for (int i = 0; i < strlen(pos); i++) { 348. temp[i] = pos[i]; 349. } 350. \*pos = '\0'; 351. *// 插入If-Modified-Since字段* 352. while (\*newfield != '\0') { 353. \*pos++ = \*newfield++; 354. } 355. *// 插入对象文件的最新被修改时间* 356. while (\*datestring != '\0') { 357. \*pos++ = \*datestring++; 358. } 359. \*pos++ = '\r'; 360. \*pos++ = '\n'; 361. for (int i = 0; i < strlen(temp); i++) { 362. \*pos++ = temp[i]; 363. } 364. } 365. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 366. *// Method: ConnectToServer* 367. *// FullName: ConnectToServer* 368. *// Access: public* 369. *// Returns: BOOL* 370. *// Qualifier: 根据主机创建目标服务器套接字，并连接* 371. *// Parameter: SOCKET \* serverSocket* 372. *// Parameter: char \* host* 373. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 374. BOOL ConnectToServer(SOCKET\* serverSocket, char\* host) { 375. sockaddr\_in serverAddr; 376. serverAddr.sin\_family = AF\_INET; 377. *// hton: 将一个16位整数从主机字节顺序转换为网络字节顺序* 378. serverAddr.sin\_port = htons(HTTP\_PORT); 379. *// HOSTENT: 处理主机信息结构体* 380. *// gethostbyname: 获取主机名对应的IP地址* 381. HOSTENT\* hostent = gethostbyname(host); 382. if (!hostent) { 383. return FALSE; 384. } 385. *// 从hostent结构中获取目标服务器的IP地址，* 386. in\_addr Inaddr = \*((in\_addr\*)\*hostent->h\_addr\_list); 387. *// 并将其转换为网络字节序的IPv4地址* 388. serverAddr.sin\_addr.s\_addr = inet\_addr(inet\_ntoa(Inaddr)); 389. *// 连接目标服务器* 390. \*serverSocket = socket(AF\_INET, SOCK\_STREAM, 0); 391. if (\*serverSocket == INVALID\_SOCKET) { 392. return FALSE; 393. } 394. if (connect(\*serverSocket, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) 395. == SOCKET\_ERROR) { 396. closesocket(\*serverSocket); 397. return FALSE; 398. } 399. return TRUE; 400. } 401. *// 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* 402. void checkfileCache(char\* buffer, char\* filename) { 403. char\* p, \* ptr, tempBuffer[MAXSIZE + 1]; 404. const char\* delim = "\r\n"; 405. ZeroMemory(tempBuffer, MAXSIZE + 1); 406. memcpy(tempBuffer, buffer, strlen(buffer)); 407. p = strtok\_s(tempBuffer, delim, &ptr); *//提取状态码所在行* 408. *//主机返回的报文中的状态码为304时返回已缓存的内容* 409. if (strstr(p, "304") != NULL) { 410. printf("\n从代理服务器获得缓存\n"); 411. ZeroMemory(buffer, strlen(buffer)); 412. FILE\* in = NULL; 413. if ((fopen\_s(&in, filename, "r")) == 0) { 414. fread(buffer, sizeof(char), MAXSIZE, in); 415. fclose(in); 416. } 417. needCache = FALSE; 418. } 419. } 420. *// 检测主机返回的状态码，如果是200则本地获取或更新缓存* 421. void storefileCache(char\* buffer, char\* url) { 422. char\* p, \* ptr, tempBuffer[MAXSIZE + 1]; 423. *//num中是状态码* 424. const char\* delim = "\r\n"; 425. ZeroMemory(tempBuffer, MAXSIZE + 1); 426. memcpy(tempBuffer, buffer, strlen(buffer)); 427. p = strtok\_s(tempBuffer, delim, &ptr); *//提取第一行* 428. *//printf("tempbuffer = %s\n", p);* 429. if (strstr(tempBuffer, "200") != NULL) { *//状态码是200时缓存* 430. char filename[100] = { 0 }; 431. makeFilename(url, filename); 432. printf("缓存文件 : %s\n", filename); 433. FILE\* out; 434. fopen\_s(&out, filename, "w+"); 435. fwrite(buffer, sizeof(char), strlen(buffer), out); 436. fclose(out); 437. printf("\n网页已经被代理服务器缓存\n"); 438. } 439. } 440. *// 黑名单用户* 441. int inUserShield(char ip[]) { 442. for (int i = 0; i < (sizeof(shieldIp) / sizeof(shieldIp[0])); i++) { 443. if (strcmp(ip, shieldIp[i]) == 0) { 444. return i; 445. } 446. } 447. return -1; 448. } 449. *// 黑名单网站* 450. int inWebShield(char ip[]) { 451. for (int i = 0; i < (sizeof(shieldWeb) / sizeof(shieldWeb[0])); i++) { 452. if (strcmp(ip, shieldWeb[i]) == 0) { 453. return i; 454. } 455. } 456. return -1; 457. } |
| 实验结果： |
| **一、使用 HTTP 代理服务器访问网页**  运行程序，代理服务器正常工作，可以转发客户端请求和服务器响应，页面显示成功，可以实现 HTTP 代理服务器基本功能。    **二、HTTP 代理服务器缓存文件**  成功访问上面网站后，本地项目文件夹中立即出现缓存文件，程序窗口中也显示“网页已经被代理服务器缓存”，且再次打开该网站时速度会快一点，可以实现缓存功能。    **三、HTTP 代理服务器网站过滤**  在网站黑名单中添加 <http://jwes.hit.edu.cn/> 网站作为不允许访问的网站，当运行程序并尝试访问该网站时，浏览器显示无法使用，没有接收到 jwes.hit.edu.cn 主机发送的数据，程序窗口也显示“该网站已被屏蔽”，可以实现网站过滤功能。    **四、HTTP 代理服务器用户过滤**  在用户黑名单中添加本机 IP 地址“120.0.0.1”作为被禁用的用户，当运行程序并尝试访问外部网站时，无法连接到服务器，且程序窗口也显示“该用户无访问权限”，可以实现用户过滤功能。    **五、HTTP 代理服务器网站引导（钓鱼）**  在代码中设置目标网站 <http://www.baidu.com> 和钓鱼网站 <http://szsj.hit.edu.cn> ，当运行程序并尝试访问目标网站时，页面自动跳转到钓鱼网站，可以实现网站引导功能。 |
| 问题讨论： |
| **一、实验结果分析**  功能实现包含了所有的实验内容，完成了实验要求的全部任务，可以实现一个具有本地缓存、网站过滤、用户过滤、网站引导功能的 HTTP 代理服务器，用于转发客户端请求和服务器响应。  **二、思考问题**  **1. Socket 编程的客户端和服务器端主要步骤**  （1）客户端：  创建 Socket：调用 socket() 函数创建一个 Socket ，用于向服务器发起连接请求。  发起连接请求：客户端调用 connect() 方法，指定服务器的地址和端口号，向服务器发起连接请求。  通信：一旦连接建立，客户端就可以使用 Socket 与服务器进行数据的发送和接收。  关闭 Socket 连接：当通信完成后，客户端需要关闭与服务器的 Socket 连接，并释放相关资源。  （2）服务器端：  创建 Socket：服务器创建一个用于监听的 Socket，绑定至服务器的本地地址和端口，用于接收来自客户端的连接请求。  创建监听：调用 listen() 函数，将 Socket 设置为监听模式，等待客户端的连接请求。  接受连接请求：服务器通过调用 accept() 方法来接受客户端的连接请求。一旦有客户端发起连接，accept() 方法会返回一个新的 Socket，这个套接字将用于与客户端进行通信。  通信：使用返回的 Socket 与客户端进行数据的发送和接收。  关闭 Socket 连接：当通信完成后，服务器需要关闭与客户端的 Socket 连接，并释放相关资源。  **2. HTTP 代理服务器的基本原理**  （1）接收请求  HTTP 代理服务器首先接收来自客户端的 HTTP 请求，通常包括请求行、请求头和请求体（对于 POST 等包含数据的请求）。客户端在发起请求时，会指明目标服务器的地址和端口，以及请求的资源路径和参数。  （2）转发请求  代理服务器在接收到客户端的请求后，会根据请求中的目标服务器地址和端口，将请求转发给相应的服务器。这个过程中，代理服务器会充当客户端和目标服务器之间的通信桥梁，确保请求能够顺利到达目标服务器。  （3）处理请求  在转发请求之前或之后，代理服务器还可以对请求进行一定的处理，包括：  a) 修改请求：代理服务器可以根据需要对原始请求进行修改，如更改请求头、添加或删除参数等；  b) 缓存：如果请求的资源已经被代理服务器缓存过，那么代理服务器可以直接从缓存中返回资源，而无需再次向目标服务器发起请求，可以大大提高访问速度和效率；  c) 访问控制：代理服务器可以根据配置的规则对请求进行访问控制，如限制对某些网站的访问、过滤不良内容等。  （4）响应请求  目标服务器在接收到代理服务器转发的请求后，会对请求进行处理并生成响应，响应通常包括状态行、响应头和响应体（对于包含数据的响应）。代理服务器在接收到目标服务器的响应后，会将其转发给客户端。 |
| 心得体会： |
| （1）掌握了Socket编程的基本流程和主要步骤，对Socket编程产生浓厚兴趣；  （2）了解了 HTTP 代理服务器的基本原理，学会了本地 Cache、用户过滤、网站过滤网站引导等功能的实现，对应用层 HTTP 请求和响应有了更深的理解；  （3）对于应用层的多线程使用有了一定了解。 |