

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | HTTP 代理服务器设计与实现 | | | | | |
| 姓名 | 何栩晟 | | 院系 | 计算机科学与技术 | | |
| 班级 | 2103101 | | 学号 | 2021113634 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | 格物213 | | 实验时间 | 2023.10.16 | | |
| 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： |
| 本次实验的主要目的： 1、 熟悉并掌握 Socket ⽹络编程的过程与技术；  2、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。  (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： |
| 1. 浏览器使用代理   这里使用SwitchyOmega插件快速设置，以便于实验过程中的调试。    如图，设置代理服务器为127.0.0.1，代理端口为10240。   1. 多线程使用   本实验中多线程通过库函数process.h实现，通过函数\_beginthreadex 创建⼦线程，使⽤函数\_endthreadex 结束线程，具体注释见下面程序。   1. ProxyParam \*lpProxyParam; 2. HANDLE hThread; 3. DWORD dwThreadId; 4. while (true) { 5. cacheFlag = false; 6. needCache = true; 7. accepetSocket = accept(proxyServer, NULL, NULL); // 接受连接 8. lpProxyParam = new ProxyParam; // 为线程参数分配内存 9. if (lpProxyParam == NULL) { 10. std::cerr << "创建线程参数失败！" << std::endl; 11. continue; 12. } 13. lpProxyParam->setClientSocket(accepetSocket); 14. hThread = (HANDLE)\_beginthreadex(NULL, 0, 15. &ProxyServer::proxyThread, (LPVOID)lpProxyParam, 0, 0); 16. // \_beginthreadex()函数用于创建一个线程，该函数的第一个参数为线程安全属性，第二个参数为堆栈大小，第三个参数为线程函数，第四个参数为传递给线程函数的参数，第五个参数为线程创建后的初始状态，第六个参数为线程ID 17. // 用到了库函数process.h 18. CloseHandle(hThread); // 关闭线程句柄 19. Sleep(200); 20. } 21. Socket创建   通过C++构造函数，在创建服务器时，就创建一个主Socket，并绑定到10240端口上，并且开始监听，等待客户机的请求。每收到一个客户机的HTTP请求，就创建一个客户机socket (clientSocket)，并利用clientSocket创建socket处理子线程。在子线程中处理其HTTP请求消息   1. ProxyServer::ProxyServer() { 2. // 初始化Winsock 3. WSADATA wsaData; 4. WSAStartup(MAKEWORD(2, 2), &wsaData); 6. // 初始化代理服务器套接字 7. proxyServer = socket(AF\_INET, SOCK\_STREAM, 0); 9. proxyServerAddr.sin\_family = AF\_INET; // IPv4 10. proxyServerAddr.sin\_addr.S\_un.S\_addr = INADDR\_ANY; // 任意地址 11. proxyServerAddr.sin\_port = htons(proxyServerPort); // 端口，转换为网络字节序 12. } 14. bool ProxyServer::initSocket() { 15. // 绑定代理服务器套接字 16. if (bind(proxyServer, (SOCKADDR \*)&proxyServerAddr, sizeof(SOCKADDR)) == SOCKET\_ERROR) { 17. std::cerr << "Failed to bind socket." << std::endl; 18. return false; 19. } 21. // 监听代理服务器套接字 22. if (listen(proxyServer, SOMAXCONN) == SOCKET\_ERROR) { 23. std::cerr << "Failed to listen socket." << std::endl; 24. return false; 25. } 27. return true; 28. } 29. 转发客户机请求   通过Winsock API的recv接受来自客户机的请求。  窗体顶端   1. recvSize = recv(((ProxyParam \*)lpParameter)->getClientSocket(), buffer, MAXSIZE, 0); // 接收数据   窗体底端  对于请求报文进行解析，处理出方法(GET or POST)，url和host。其中大量用到了strtok\_s函数，具体可见注释。   1. void ProxyServer::parseHttpHead(char \*buffer, HttpHeader \*httpHeader) { 2. char \*p; 3. char \*ptr; 4. const char \*split = "**\r\n**"; 5. p = strtok\_s(buffer, split, &ptr); // 分割字符串 6. // strtok\_s()函数用于分割字符串，第一个参数为要分割的字符串，第二个参数为分割字符串，第三个参数为保存分割结果的指针 7. if (p[0] == 'G') { // GET 8. httpHeader->setMethod("GET"); 9. httpHeader->setUrl(p + 4, strlen(p) - 13); 10. } 11. else if (p[0] == 'P') { // POST 12. httpHeader->setMethod("POST"); 13. httpHeader->setUrl(p + 5, strlen(p) - 14); 14. } 15. else { 16. // printf("暂不支持该方法！%s\n", p); 17. return; 18. } 19. p = strtok\_s(NULL, split, &ptr); // NULL表示使用上一次的分割结果 20. while (p) { 21. if (p[0] == 'H') { // Host 22. httpHeader->setHost(p + 6, strlen(p) - 6); 23. } 24. else if (p[0] == 'C') { // Cookie 25. if (strlen(p) > 8) { 26. char header[8]; 27. ZeroMemory(header, sizeof(header)); // zeromemory函数用于将内存块清零, 和memset的区别是前者是系统函数, 后者是C函数 28. memcpy(header, p, 6); 29. if (strcmp(header, "Cookie") == 0) { 30. httpHeader->setCookie(p + 8, strlen(p) - 8); 31. } 32. } 33. } 34. p = strtok\_s(NULL, split, &ptr); 35. } 36. }   进一步处理出文件名（为缓存做准备），主机名（提取ip，为用户过滤做准备）   1. void ProxyServer::getFilename(const char \*url, char \*filename, size\_t maxLen) { 2. size\_t len = strlen(url); 3. size\_t copyLen = (len < maxLen) ? len : maxLen - 1; // 考虑数组边界，保留一个位置给字符串结尾的'\0' 4. size\_t filenameIndex = 0; 6. for (size\_t i = 0; i < copyLen; i++) { 7. if (url[i] != '/' && url[i] != ':' && url[i] != '.') { // 过滤掉非法字符 8. filename[filenameIndex++] = url[i]; 9. } 10. } 12. filename[filenameIndex] = '**\0**'; // 添加字符串结尾标志 13. } 14. ... 15. getFilename(httpHeader->getUrl(), fileName, sizeof(fileName)); // 获取文件名 16. printf("文件名：%s**\n**", fileName); 18. int ret\_new = gethostname(hostname, sizeof(hostname)); // 获取主机名 19. printf("主机名：%s**\n**----------------**\n**", hostname); 21. HOSTENT \*hostent = gethostbyname(hostname); // 获取主机信息 22. char \*ip = inet\_ntoa(\*(in\_addr \*)\*hostent->h\_addr\_list); // 获取IP地址   将请求报文经过代理服务器通过API send函数发送至目标主机，通过recv函数接收来自目标主机的应答报文，过程中充分考虑错误信息的处理和输出，以便于后期调试。   1. bool ProxyServer::connectToServer(SOCKET \*serverSocket, const char \*host) { 2. sockaddr\_in serverAddr; 3. serverAddr.sin\_family = AF\_INET; 4. serverAddr.sin\_port = htons(HTTP\_PORT); // 端口，转换为网络字节序 5. HOSTENT \*hostent = gethostbyname(host); // 获取主机信息 6. if (!hostent) { 7. printf("获取主机信息失败！**\n**错误代码为%d**\n**", WSAGetLastError()); 8. return false; 9. } 10. in\_addr Inaddr = \*((in\_addr \*)\*hostent->h\_addr\_list); // 获取主机地址 11. serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr(inet\_ntoa(Inaddr)); // 点分十进制转换为网络字节序 12. \*serverSocket = socket(AF\_INET, SOCK\_STREAM, 0); // 创建套接字 13. if (INVALID\_SOCKET == \*serverSocket) { 14. printf("创建套接字失败！**\n**错误代码为%d**\n**", WSAGetLastError()); 15. return false; 16. } 17. if (connect(\*serverSocket, (SOCKADDR \*)&serverAddr, sizeof(SOCKADDR)) == SOCKET\_ERROR) { // 连接服务器 18. printf("连接服务器失败！**\n**错误代码为%d**\n**", WSAGetLastError()); 19. return false; 20. } 21. return true; 22. } 24. success: 25. SOCKET serverSockett; 26. serverSockett = ((ProxyParam \*)lpParameter)->getServerSocket(); 27. if (strlen(httpHeader->getHost()) == 0) 28. goto error; 29. if (!connectToServer(&serverSockett, httpHeader->getHost())) { 30. printf("连接服务器失败！%s **\n**", httpHeader->getHost()); 31. goto error; 32. } 33. ((ProxyParam \*)lpParameter)->setServerSocket(serverSockett); 34. if (!redirectFlag) 35. printf("连接服务器成功！**\n**请求报文: **\n**%s**\n**", buffer); 36. ret = send(((ProxyParam \*)lpParameter)->getServerSocket(), buffer, strlen(buffer) + 1, 0); // 发送数据 37. if (ret == -1) { 38. int errCode = WSAGetLastError(); 39. printf("发送数据失败，错误代码：%d**\n**", errCode); 40. } 41. recvSize = recv(((ProxyParam \*)lpParameter)->getServerSocket(), buffer, MAXSIZE, 0); // 接收数据 42. if (recvSize <= 0) { 43. printf("接收数据失败！**\n**"); 44. goto error; 45. } 46. 客户主机过滤   建立过滤主机列表，通过strcmp匹配IP地址，若是屏蔽ip则直接关闭socket，不进行下一步的操作。   1. if (!strcmp(ip, "127.0.0.1")) { 2. printf("本地访问**\n**您的主机被屏蔽**\n**"); 3. goto error; 4. } 5. 网站过滤   建立过滤网址列表，通过strstr和strcmp匹配，若是被屏蔽的网站则直接关闭socket，不进行下一步的操作。   1. if (strstr(httpHeader->getUrl(), "gov.cn")) { 2. printf("本地访问%s**\n**您的主机被屏蔽**\n**", httpHeader->getUrl()); 3. goto error; 4. } 5. 网站引导（钓鱼）   通过更改请求报文，将目标的url和host进行修改，从而实现网站的引导和钓鱼。   1. if (!strcmp(httpHeader->getHost(), "jwes.hit.edu.cn")) { 2. printf("**\n**=====================================**\n\n**"); 3. printf("钓鱼成功：您所前往的网站已被引导至http://jwts.hit.edu.cn**\n**"); 4. httpHeader->setHost("jwts.hit.edu.cn", 22); 5. httpHeader->setUrl("http://jwts.hit.edu.cn", 22); 6. redirectFlag = true; 7. } 8. 实现缓存Cache   通过fopen\_s和fwrite将得到的响应报文写入本地，通过url构造文件名，从而实现缓存功能。   1. bool writeFile(const char \*filename, const char \*data) { 2. FILE \*file; 3. if (fopen\_s(&file, filename, "wb") == 0) { // wb 以二进制写入 4. size\_t bytesWritten = fwrite(data, sizeof(char), strlen(data), file); // 写入文件 5. fclose(file); 6. return bytesWritten > 0; 7. } 8. return false; 9. } 10. void ProxyServer::saveCache(char \*buffer, const char \*url) { 11. const char \*split = "**\r\n**"; 12. char \*p, \*ptr, status[4]; 13. char tempBuffer[MAXSIZE + 1]; 15. memset(status, 0, sizeof(status)); 16. memset(tempBuffer, 0, sizeof(tempBuffer)); 17. strncpy(tempBuffer, buffer, sizeof(tempBuffer) - 1); 19. p = strtok\_s(tempBuffer, split, &ptr); // 提取第一行 20. strncpy(status, p + 9, sizeof(status) - 1); // 提取状态码 22. if (strcmp(status, "200") == 0) { 23. char filename[100]; 24. memset(filename, 0, sizeof(filename)); 25. getFilename(url, filename, sizeof(filename)); // 获取文件名 26. if (writeFile(filename, buffer)) { 27. printf("缓存成功！**\n**"); 28. } 29. else { 30. printf("缓存失败：无法写入文件。**\n**"); 31. } 32. } 33. } 34. 实现验证缓存是否最新   首先通过getDate获取缓存文件中的时间。   1. bool ProxyServer::getDate(char \*buffer, char \*date) { 2. char \*p; 3. char \*ptr; 4. const char \*split = "**\r\n**"; 5. p = strtok\_s(buffer, split, &ptr); // 分割字符串 6. while (p) { 7. if (p[0] == 'D') { // Date 8. if (strlen(p) > 6) { 9. char header[6]; 10. ZeroMemory(header, sizeof(header)); 11. memcpy(header, p, 4); 12. if (strcmp(header, "Date") == 0) { 13. memcpy(date, p + 6, strlen(p) - 6); 14. return true; 15. } 16. } 17. } 18. p = strtok\_s(NULL, split, &ptr); 19. } 20. return false; 21. }   接着需要在请求报文中加入一行If-Modified-Since:…，向目标主机确认是否是最新的缓存。在这一步要注意，首先要先扫描一遍请求报文自身是否本来就有这一行，以免出现错误。   1. void ProxyServer::changeHTTP(char \*buffer, char \*value) { 2. const char\* fieldToReplace = "Host"; 3. const char\* newField = "If-Modified-Since: "; 4. char\* pos = strstr(buffer, fieldToReplace); 5. char \*check = strstr(buffer, newField); 6. char \*pos1 = strstr(pos, "**\r\n**"); 7. if (check != NULL) { 8. char temp[MAXSIZE]; 9. strncpy(temp, buffer, pos - buffer); 10. strncat(temp, pos, pos1 - pos); 11. char\* oldHostEnd = strstr(pos + strlen(fieldToReplace), "**\r\n**"); 12. if (oldHostEnd != NULL) { 13. // 要把host后面到newField前面的内容补上 14. strncat(temp, oldHostEnd, check - oldHostEnd); 15. } 16. strcat(temp, newField); 17. strcat(temp, value); 18. strcat(temp, "**\r\n**"); 19. // 要把 newField 后面的内容补上 20. char \*oldHostEnd1 = strstr(check + strlen(newField), "**\r\n**"); 21. if (oldHostEnd1 != NULL) { 22. strcat(temp, oldHostEnd1); 23. } 24. ZeroMemory(buffer, strlen(buffer)); 25. memcpy(buffer, temp, strlen(temp)); 26. } 27. else if (pos != NULL) { 28. char temp[MAXSIZE]; 29. strncpy(temp, buffer, pos - buffer); 30. strcat(temp, newField); 31. strcat(temp, value); 32. strcat(temp, "**\r\n**"); 33. // 要把 host 那行补上 34. strncat(temp, pos, pos1 - pos); 36. char\* oldHostEnd = strstr(pos + strlen(fieldToReplace), "**\r\n**"); 37. if (oldHostEnd != NULL) { 38. strcat(temp, oldHostEnd); 39. } 40. ZeroMemory(buffer, strlen(buffer)); 41. memcpy(buffer, temp, strlen(temp)); 42. } 43. } 44. 关闭套接字   一个子线程结束要关闭套接字。   1. error: 2. Sleep(200); 3. closesocket(((ProxyParam \*)lpParameter)->getClientSocket()); 4. closesocket(((ProxyParam \*)lpParameter)->getServerSocket()); 5. delete (ProxyParam \*)lpParameter; 6. \_endthreadex(0); 7. return 0; |
| 实验结果： |
| 1. 基础功能   如下图所示，可以正常通过代理服务器获取网页内容，并在控制台打印相关信息。     1. Cache缓存   jwts.hit.edu.cn经测试，每次都会更新缓存，因此这里采用<http://info.cern.ch/hypertext/WWW/TheProject.html>。  第一次访问，本地没有缓存，代理服务器在本地没有缓存，因此正常转发客户机的请求报文，可以看到此时还没有添加If-Modified-Since:。    对响应报文进行缓存，如下图所示。此时状态码为200。    第二次访问时，看到控制台显示**缓存命中**，并且看到请求报文中添加了If-Modified-Since:行来询问是否是最新的缓存。此时状态码变为304 Not Modified，说明缓存确实是最新的。     1. 网站过滤   过滤列表为禁止访问政府网站，即以gov.cn结尾的网站。       1. IP 过滤   这里可以对127.0.0.1进行过滤，禁止访问一切网页。     1. 网站引导   这里引导若访问jwes.hit.edu.cn将会被钓鱼至jwts.hit.edu.cn |
| 问题讨论： |
| * 1. Socket 编程的客户端和服务器端主要步骤     客户端：   1. 通过WSAStartup函数初始化加载套接字库 2. 创建一个socket，用于与服务器端连接，socket(AF\_INET,SOCK\_STREAM)方法创建套接字，AF\_INET表示ipv4，SOCK\_STREAM表示是面向TCP连接的流式套接字。 3. 用connect()方法与服务器端连接 4. 利用套接字的send()方法向服务器端发送请求消息，返回值为成功发送的字节数，该值可能会小于需要发送的字节数。 5. 发送完请求消息后，开始处于等待状态，当服务器端返回数据到达时，利用recv()函数接受数据，返回的类型为字符串形式，其中还可以规定接受的最大字节数。 6. 使用close函数关闭socket连接。 7. 使用WSACleanup函数释放所使用的Windows Sockets DLL。   服务器端：   1. 通过WSAStartup函数初始化加载套接字库。 2. 创建一个socket，方式同客户端。 3. 利用bind，绑定socket到一个IP地址和端口号。 4. 使用listen()函数进行监听，等待客户端的连接请求。 5. 使用accpet()接受客户端的连接请求，建立一个新的socket用于与客户端通信。 6. 通过新建立的socket与客户端进行通信，调用recv接收和sned发送数据。 7. 调用close()关闭连接。 8. 调用WSACleanup函数释放所使用的Windows Sockets DLL。    1. HTTP代理服务器的基本原理   HTTP代理服务器是一种特殊的网络服务，它允许客户端通过它来访问互联网。当客户端向代理服务器发送HTTP请求时，代理服务器会将该请求转发到目标服务器，并将目标服务器的响应返回给客户端。HTTP代理服务器的基本原理如下：   1. 客户端向HTTP代理服务器发送HTTP请求。 2. HTTP代理服务器接收到客户端的请求后，解析请求中的URL，确定目标服务器的IP地址和端口号。 3. HTTP代理服务器向目标服务器发送HTTP请求，请求中包含客户端的请求信息。 4. 目标服务器接收到HTTP代理服务器的请求后，处理该请求并返回响应。 5. HTTP代理服务器接收到目标服务器的响应后，将响应转发给客户端。      * 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  1. 网络编程：HTTP代理服务器需要使用网络编程技术来接收客户端的连接请求，与客户端和目标服务器进行通信。使用Socket编程来实现这一点。 2. HTTP协议解析：HTTP代理服务器需要能够解析客户端发送的HTTP请求，提取出请求中的URL、方法、头部等信息。使用拆解和解析报文各个字段来完成。 3. 多线程或异步编程：为了提高性能，HTTP代理服务器需要能够同时处理多个客户端的请求。 |
| 心得体会： |
| 1. 对socket编程的基本函数和基本操作方法有了初步的了解，本次实验是在windows下完成，之后想在linux下也写一次。 2. 对于TCP、HTTP服务器的基本原理有了更深入的了解，掌握了代理服务器的基本实现过程。 3. 对拓展HTTP代理服务器：网站钓鱼、网站屏蔽、用户屏蔽的简单实现机制有了深刻的认识和理解； |