

**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 | 阎发祥 | | 院系 | 计算学部 | | |
| 班级 | 2203101 | | 学号 | 2023140004 | | |
| 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。 |
| 实验环境： |
| 接入 Internet 的实验主机  Windows 11, Visual Studio 2022  开发语言：C语言 |
| 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.（选做）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相应的对象，并能够通过修改请求报文(添加if-modified-since头部行)，向原服务器确认缓存对象是否是最新版本。  3.（选做）扩展HTTP代理服务器，使其能够  （a）支持网站过滤，允许/不允许访问某些网站。  (b)支持用户过滤，支持/不支持某些用户访问外部网站。  (c)支持网站引导，将用户对某个网站的访问引导至一个模拟网站(钓鱼)。 |
| 实验原理： |
| 1. Socket编程的客户端和服务器端的主要步骤   （1）socket客户端  编程的思路是明确目的服务器的IP地址、端口号以及传输层协议(TCP or UDP)，首先使用 socket 函数创建一个 socket 描述符，来唯一标识一个 socket；然后使用 bind 函数绑定 IP 地址，端口信息；然后使用 listen 函数进行监听创建的 socket；接着使用 accept 函数接收请求，此时建立socket连接；然后使用 read(), write() 等函数调用网络I/O 进行读写操作实现通信；最后接收完消息使用 close 函数关闭网络连接  （2）socket服务器端  对于socket服务器端而言，主要区分一下TCP协议和UDP协议上的编程即可。对于UDP协议上的通信，无需提前建立连接，只需在开始时建立相应的socket，接收消息后直接与源地址进行通信。对于TCP协议上的通信，大体思路是创建 socket，绑定 IP 地址，端口信息，设置要连接对方的 IP 地址和端口属性，使用connect 函数连接服务器，使用 read(), write()等函数进行网络 I/O 的读写，最后关闭网络连接  流程图如下（参考mooc课程）    2. HTTP 代理服务器的基本原理  我们在浏览器和服务器请求与响应的中间加入一层，即浏览器向代理服务器发送请求，代理服务器去解析这个请求，然后向真正的服务器去发送请求，然后代理服务器得到真正服务器所得到的响应，然后再发送给客户端。  代理服务器相对于客户端来说就是服务器；相对于真正的服务器来说，代理服务器就是客户端。代理服务器在整个过程中类似于中介。  下图是Web通信方式的对比      3.HTTP代理服务器的程序流程图  4. 实现 HTTP 代理服务器的关键技术及解决方案  （a）程序基本功能：实现请求报文和相应报文的转发   1. 初始化套接字，利用 bind() 函数将该套接字与服务器 host 地址绑定，地址设为 “127.0.0.1”；同时绑定端口号为“10240”。在设置完毕后，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，为提高效率，对每个请求，代理服务器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。 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。   （b）解析HTTP请求报文技术描述  解析HTTP的请求报文主要是解析HTTP的头部行，头部行标注着method、URL和协议的版本号，并使用1个空格进行划分    （c）拓展功能：网站过滤、用户过滤、网站钓鱼  设置被屏蔽的网站为哈工大网站（http://hit.edu.cn/），设置钓鱼网站为北航官网（http://www.buaa.edu.cn）重定向至哈工大本科生服务平台（http://jwes.hit.edu.cn/），相关示例网站可以更换  在线程执行函数ProxyThread中，解析TCP报文中的HTTP头部，将HTTP头部中的URL与对应屏蔽网站URL BANNED\_WEB进行比较，如果相同，则打印相关语句并直接跳转到error，从而实现网站过滤功能。用户过滤功能与其类似，不过是在监听过程中实现。网站钓鱼将其中的IP地址修改为钓鱼目的IP地址即可实现功能，相关函数如图所示      （d）拓展功能：Cache功能实现  对于cache功能的实现，思路是将所有的请求的文件保存在磁盘上。代理服务器第一次和客户端通信时，会保留Cache；当源主机再次访问相同的文件时，proxy首先获得缓存文件的时间，然后构造条件GET方法(增加if-modified-since头部)访问目的服务器，如果状态码为200，则表示内容不是最新的，则由目标服务器返回新的内容，如果内容未经修改，仍是最新的，则返回 304 告诉客户端其本地 cache 的页面是最新的，于是代理服务器可以本地Cache的发送给客户端。 |
| 实验结果： |
| 1. 对于代理服务器的基本功能实现，访问http://jwes.hit.edu.cn，访问的结果如下：       1. 网页限制        1. 用户限制   修改屏蔽用户ip为127.0.0.1，运行程序如下    4.钓鱼的实现  输入北航官网，得到的是哈工大本科生服务平台，表明成功钓鱼      5.cache的实现  连续访问两次<http://jwes.hit.edu.cn/>，在文件夹中会产生缓存文件    再次访问时，服务器直接从缓存中读取文件，节约时间 |
| 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://www.hit.edu.cn/";//不允许访问的网站    char Target\_web[1024] = "http://www.buaa.edu.cn";//钓鱼原网站  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    char InvalidIP[] = "127.0.0.2";//屏蔽的用户IP  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  int addrlen = 32;    //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  SOCKADDR\_IN acceptAddr; //自定义变量，用来获得用户的IP  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, (SOCKADDR\*)&acceptAddr, &addrlen);  printf("===============用户IP地址为%s===============\n", inet\_ntoa(acceptAddr.sin\_addr));  if (strcmp(inet\_ntoa(acceptAddr.sin\_addr), InvalidIP) == 0)//屏蔽用户ip  {  printf("\n\n===============该用户已经被屏蔽===============\n\n");  }  else {  lpProxyParam = new ProxyParam;//每个请求都创建一个新的线程来处理  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 = 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); //解析HTTP报文头部  //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\n", 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\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);  fread(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");  }  }    //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");  ZeroMemory(buffer, strlen(buffer));  FILE\* in = NULL;  if ((fopen\_s(&in, filename, "r")) == 0) {  fread(buffer, sizeof(char), MAXSIZE, in);  fclose(in);  }  needCache = FALSE;  }  } |