



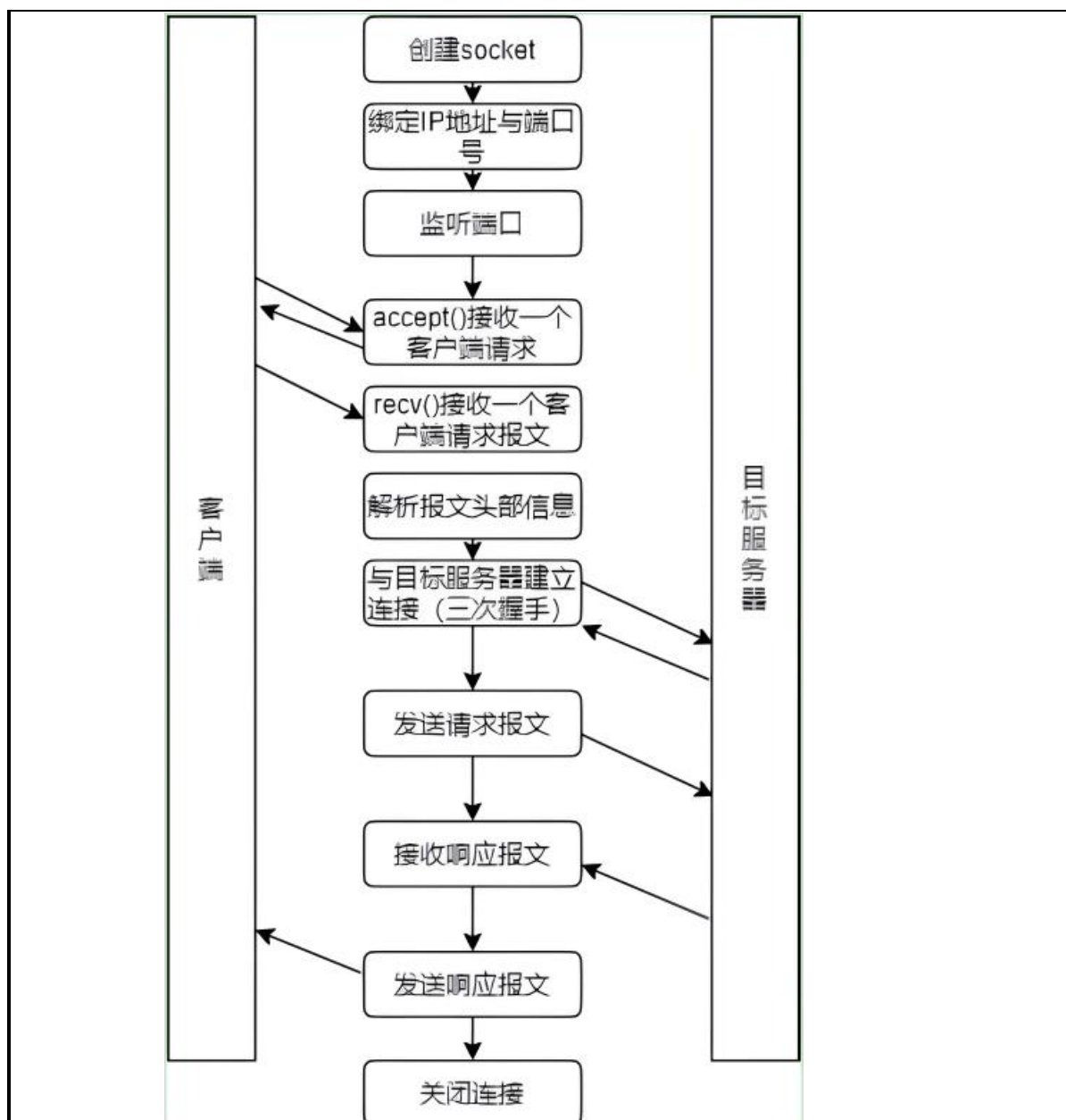
哈尔滨工业大学
Harbin Institute of Technology

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		院系	计算学部			
班级		学号				
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：
<p>(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。</p> <p>(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。</p> <p>(3)扩展HTTP代理服务器，支持如下功能：</p> <p>a)网站过滤：允许/不允许访问某些网站；</p> <p>b)用户过滤：支持/不支持某些用户访问外部网站；</p> <p>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）</p>
实验过程：
<p>一. Socket 编程的客户端和服务端主要步骤</p> <p>客户端：</p> <ol style="list-style-type: none"> 1. 根据目的服务器IP地址和端口号创建套接字，连接服务器 2. 发送请求报文 3. 接受返回报文 4. 关闭连接 <p>服务器端：</p> <ol style="list-style-type: none"> 1. 创建套接字，绑定IP地址和端口号，监听端口 2. 从连接队列中取出一个连接请求，三次握手创建连接 3. 接收请求报文 4. 发送响应报文 5. 关闭当前连接，继续监听端口 <p>二. HTTP 代理服务器的基本原理</p> <p>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）进行非直接连接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。</p> <p>三. HTTP 代理服务器的程序流程图</p>



四. 实现HTTP 代理服务器的关键技术及解决方案

4.1 基本 HTTP 代理服务器：设计并实现一个基本 HTTP 代理服务器，能够接收来自客户的 HTTP 请求，并根据其中的 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。

设计思路：使用多线程模型，为每个客户端创建一个代理线程，代理线程负责与客户端和原服务器进行通信。代理线程首先从客户端套接字接收 HTTP 请求报文，并解析其中的 URL 地址和主机名。然后，代理线程创建一个与原服务器通信的套接字，并根据主机名连接到原服务器。如果连接成功，代理线程将 HTTP 请求报文转发给原服务器，并从原服务器套接字接收 HTTP 响应报文。最后，代理线程将 HTTP 响应报文转发给客户端，并关闭与客户端和原服务器的套接字。

实现步骤：

1. 创建一个监听套接字，绑定到指定端口（例如 8080），并开始监听客户端的连接请求。
2. 当有客户端连接请求到来时，接受连接请求，并创建一个客户端套接字。
3. 创建一个代理参数结构体，用于存储客户端套接字和原服务器套接字。
4. 创建一个代理线程，并将代理参数结构体作为参数传递给代理线程。

在代理线程中，执行以下操作：

定义一个缓冲区数组，用于存储 HTTP 报文数据。

定义一个 `HttpHeader` 结构体，用于存储 HTTP 请求头中的方法、URL、主机名和 Cookie 等信息。

```
//用于存储 HTTP 请求头的信息
struct HttpHeader {
    char method[4]; // 存储 HTTP 方法，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() { // 构造函数，初始化结构体的成员变量
        ZeroMemory(this, sizeof(HttpHeader)); // 将结构体的内存空间清零
    }
};
```

从客户端套接字接收 HTTP 请求报文，并将其存储到缓冲区数组中。

调用 `ParseHttpHead` 函数，解析缓冲区数组中的 HTTP 请求头，并将结果存储到 `HttpHeader` 结构体中。

```
// Qualifier: 解析 TCP 报文中的 HTTP 头部
void ParseHttpHead(char* buffer, HttpHeader* httpHeader) {
    char* p; //指向分割后的字符串
    char* ptr; //存储分割函数的上下文信息
    const char* delim = "\r\n";

    p = strtok_s(buffer, delim, &ptr); //调用 strtok_s 函数从 buffer 指向的字符串中分割出第一个子字符串

    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);
    }
}
```

p = strtok_s(NULL, delim, &ptr); //调用 strtok_s 函数从 buffer 指向的字符串中继续分割出下一个子字符串

```
while (p) {
    switch (p[0]) {
        case 'H': // 表示 HTTP 请求头中的主机名字段
            memcpy(httpHeader->host, &p[6], strlen(p) - 6);
            break;
        case 'C': // 表示 HTTP 请求头中可能有 Cookie 字段
            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);
}
}
```

调用 ConnectToServer 函数，根据 HttpHeader 结构体中的主机名创建并连 5. 接到原服务器套接字。如果连接失败，则跳转到错误处理部分。

```
// Qualifier: 根据主机创建目标服务器套接字，并连接
BOOL ConnectToServer(SOCKET* serverSocket, char* host) {
    sockaddr_in serverAddr; // 存储目标服务器的地址信息
    serverAddr.sin_family = AF_INET; // IPv4 协议
    serverAddr.sin_port = htons(HTTP_PORT); // 使用 HTTP 端口号
    HOSTENT* hostent = gethostbyname(host); // 主机名字符串
    if (!hostent) {
        return FALSE;
    }
    in_addr Inaddr = *((in_addr*)*hostent->h_addr_list); // 获取目标服务器的 IP 地址
    serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr)); // 设置目标服务器的 IP 地址
    *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;
}

```

将缓冲区数组中的 HTTP 请求报文发送给原服务器套接字。

从原服务器套接字接收 HTTP 响应报文，并将其存储到缓冲区数组中。

将缓冲区数组中的 HTTP 响应报文发送给客户端套接字。

跳转到错误处理部分。

6. 在错误处理部分，执行以下操作：

关闭与客户端和原服务器的套接字。

释放代理参数结构体占用的内存空间。

结束代理线程并返回 0。

4.2 支持 Cache 功能的 HTTP 代理服务器：在基本 HTTP 代理服务器的基础上，增加 Cache 功能，能够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。

设计思路：使用文件系统作为缓存存储空间，为每个缓存对象创建一个文件，并根据 URL 地址生成文件名。在代理线程中，首先判断是否有缓存文件可用，如果有，则读取缓存文件中的 HTTP 响应头，并提取其中的日期字段。然后，修改 HTTP 请求报文，添加 if-modified-since 头行，并将日期字段作为值。接着，将修改后的 HTTP 请求报文转发给原服务器，并接收原服务器的 HTTP 响应报文。如果原服务器返回的状态码为 304，表示缓存对象是最新版本，则从缓存文件中读取 HTTP 响应内容，并覆盖缓冲区数组中的数据。如果原服务器返回的状态码为 200，表示缓存对象已经过期，则将缓冲区数组中的数据写入缓存文件中，并更新缓存文件。最后，将缓冲区数组中的数据转发给客户端。

实现步骤：

1. 在代理线程中，在从客户端套接字接收 HTTP 请求报文之后，执行以下操作：

定义一个布尔变量 haveCache，用于表示是否有缓存文件可用。

定义一个布尔变量 needCache，用于表示是否需要缓存文件。

定义一个字符数组 filename，用于存储缓存文件名。

调用 makeFilename 函数，根据 HttpHeader 结构体中的 URL 生成缓存文件名，并将结果存储到 filename 数组中。

```

//根据 url 构造文件名
void makeFilename(char* url, char* filename) {

```

```
while (*url != '\0') {  
    if (*url != '/' && *url != ':' && *url != '.') {  
        *filename++ = *url; //将 url 指向的字符复制到 filename 指向的位置  
    }  
    url++;  
}  
strcat(filename, ".txt");  
}
```

判断 filename 数组是否为空，如果不为空，则执行以下操作：

2. 判断 filename 对应的文件是否存在，如果存在，则执行以下操作：

将 haveCache 变量设置为 TRUE。

定义一个字符数组 tempDate，用于存储日期字段。

调用 ParseDate 函数，从 filename 对应的文件中读取 HTTP 响应头，并提取其中的日期字段，并将结果存储到 tempDate 数组中。

调用 makeNewHTTP 函数，修改缓冲区数组中的 HTTP 请求报文，添加 if-modified-since 头行，并将 tempDate 数组作为值。

否则，将 needCache 变量设置为 TRUE。

3. 在从原服务器套接字接收 HTTP 响应报文之后，执行以下操作：

如果 haveCache 变量为 TRUE，则执行以下操作：

4. 调用 getCache 函数，从 filename 对应的文件中读取 HTTP 响应内容，并覆盖缓冲区数组中的数据。

将 needCache 变量设置为 FALSE。

5. 如果 needCache 变量为 TRUE，则执行以下操作：

调用 makeCache 函数，将缓冲区数组中的数据写入 filename 对应的文件中，并创建或更新缓存文件。

4.3 扩展功能：在支持 Cache 功能的 HTTP 代理服务器的基础上，增加如下功能：

4.3.1 网站过滤：允许/不允许访问某些网站；

设计思路：使用一个字符指针数组 ForbiddenUrl，用于存储被禁止访问的网站的 URL 地址。在代理线程中，在解析 HTTP 请求头之后，调用 ForbiddenToConnect 函数，判断 HttpHeader 结构体中的 URL 是否在 ForbiddenUrl 数组中。如果是，则打印不允许访问的信息，并跳转到错误处理部分。

实现步骤：

```
//Qualifier:实现网站过滤, 不允许访问某些网站
bool ForbiddenToConnect(char* httpheader)
{
    char* forbiddernUrl = (char*)"http://www.hit.edu.cn/";
    if (!strcmp(httpheader, forbiddernUrl))
    {
        return true;
    }
    else
        return false;
}
```

定义一个字符指针数组 ForbiddenUrl, 初始化为一些被禁止访问的网站的 URL 地址。

在代理线程中, 在解析 HTTP 请求头之后, 执行以下操作:

调用 ForbiddenToConnect 函数, 判断 HttpHeader 结构体中的 URL 是否在 ForbiddenUrl 数组中。参数分别表示 URL 字符串和 ForbiddenUrl 数组。

如果返回值为 true, 则执行以下操作:

打印不允许访问的信息到控制台。

跳转到错误处理部分。

4.3.2 用户过滤: 支持/不支持某些用户访问外部网站;

设计思路: 使用一个字符指针数组 ForbiddenIP, 用于存储被禁止访问代理服务器的用户的 IP 地址。在代理线程中, 在接收客户端套接字之后, 调用 UserIsForbidden 函数, 判断客户端套接字的 IP 地址是否在 ForbiddenIP 数组中。如果是, 则打印不支持的信息, 并跳转到错误处理部分。

实现步骤:

```
//Qualifier:实现用户过滤, 禁用IP
bool UserIsForbidden(char* userID)
{
    for (int i = 0; i < IPnum; i++)// 遍历被禁止访问代理服务器的 IP 地址数组
    {
        if (strcmp(userID, ForbiddenIP[i]) == 0)
        {
            //用户IP在禁用IP表中
            return true;
        }
    }
    return false;
}
```

定义一个字符指针数组 ForbiddenIP, 初始化为一些被禁止访问代理服务器的用户的 IP 地址。

在代理线程中, 在接收客户端套接字之后, 执行以下操作:

调用 UserIsForbidden 函数, 判断客户端套接字的 IP 地址是否在 ForbiddenIP 数组中。参数分别表示 IP 地址字符串和 ForbiddenIP 数组。

如果返回值为 true，则执行以下操作：

打印不支持的信息到控制台。

跳转到错误处理部分。

4.3.3 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）；

设计思路：使用一个字符指针数组 fishUrl，用于存储需要跳转到钓鱼网站的网站的 URL 地址。在代理线程中，在解析 HTTP 请求头之后，调用 GotoFalseWebsite 函数，判断 HttpHeader 结构体中的 URL 是否在 fishUrl 数组中。如果是，则创建一个钓鱼网站响应报文，并将其存储到 FishBuffer 数组中。然后，将 FishBuffer 数组中的数据发送给客户端，并跳转到错误处理部分。

实现步骤：

```
//Qualifier:实现访问引导到模拟网站
bool GotoFalseWebsite(char* url)
{
    cout << url << endl;
    for (int i = 0; i < fishUrlnum; i++)// 从 0 到 fishUrlnum - 1 遍历钓鱼网站 URL 数组
    {
        if (strcmp(url, fishUrl[i]) == 0)
        {
            return true;
        }
    }
    return false;
}
```

定义一个字符指针数组 fishUrl，初始化为一些需要跳转到钓鱼网站的网站的 URL 地址。在代理线程中，在解析 HTTP 请求头之后，执行以下操作：

调用 GotoFalseWebsite 函数，判断 HttpHeader 结构体中的 URL 是否在 fishUrl 数组中。参数分别表示 URL 字符串和 fishUrl 数组。

如果返回值为 true，则执行以下操作：

定义一个字符数组 FishBuffer，用于存储钓鱼网站响应报文。

定义一个字符指针变量 pr，用于指向 FishBuffer 数组中的当前位置。

定义一个整型变量 fishing_len，用于表示钓鱼网站响应数据的长度。

使用 memcpy 函数和 strlen 函数，将一些固定的字符串复制到 FishBuffer 数组中，如 "HTTP/1.1 302 Moved Temporarily\r\n"、"Connection:keep-alive\r\n"、"Cache-Control:max-age=0\r\n" 等，并更新 pr 变量和 fishing_len 变量。

使用 memcpy 函数和 strlen 函数，将一个指向钓鱼网站的字符串复制到 FishBuffer 数组中，如 "Location: <http://today.hit.edu.cn/>\r\n\r\n"，并更新 pr 变量和 fishing_len 变量。

调用 send 函数，将 FishBuffer 数组中的数据发送给客户端套接字，并将返回值赋值给 ret 变量。参数分别表示客户端套接字、发送缓冲区、缓冲区大小和标志位。

跳转到错误处理部分。

实验结果：

1. 基本代理服务器的实现

编辑代理服务器

使用代理服务器

☒ 开

代理 IP 地址

127.0.0.1

端口

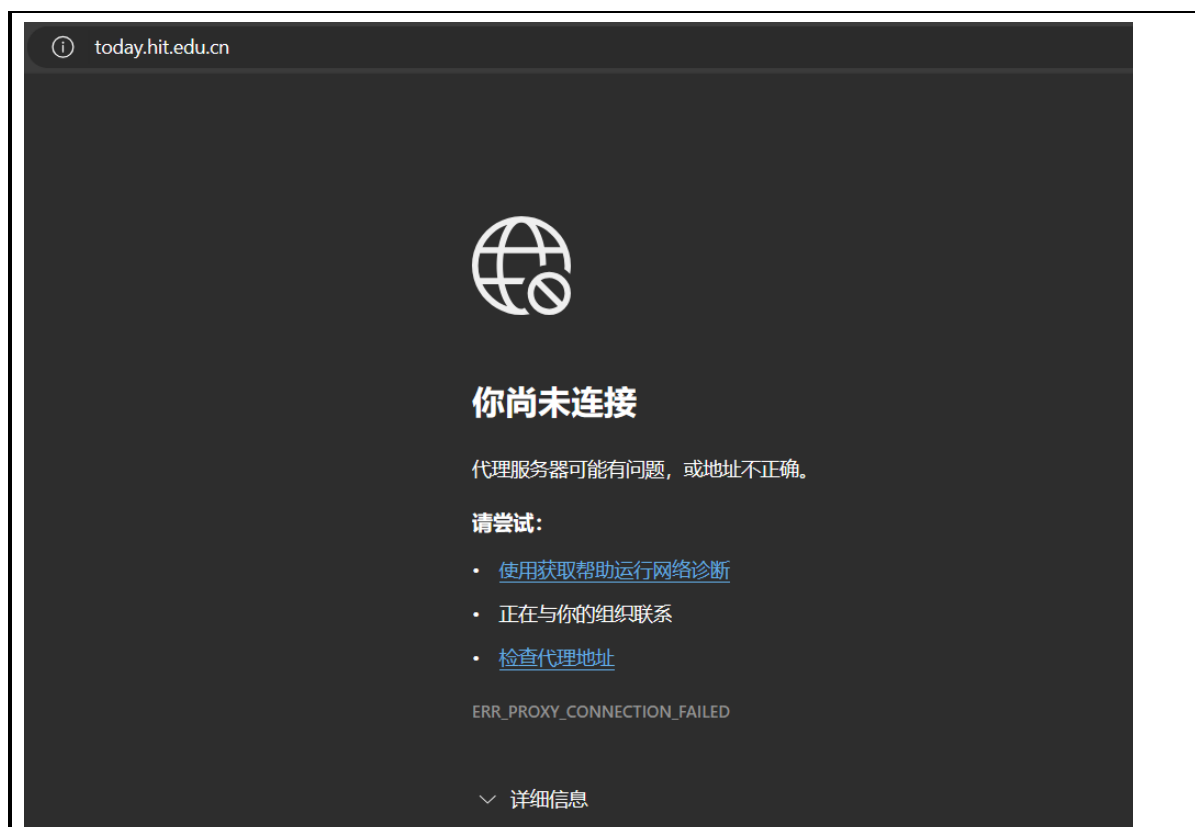
10240

请勿对以下列条目开头的地址使用代理服务器。若有多个条目，请使用英文分号 (;) 来分隔。

localhost;127.*;192.168.*

☒ 请勿将代理服务器用于本地(Intranet)地址

首先我们先手动设置代理服务器，IP地址为127.0.0.1，端口设为10240，与程序中监听端口保持一致



在未运行程序时，无法访问http://today.hit.edu.cn网站。



```
代理连接主机 today.hit.edu.cn 成功
http://today.hit.edu.cn/
代理连接主机 today.hit.edu.cn 成功
filename : httptodayhiteducn.txt
```

当程序执行时，代理服务器便正常工作，能将数据报文发给客户端，最终页面显示成功。至此，基本代理服务器的功能便实现完成。

2. 设计支持Cache功能的HTTP代理服务器

当我们打开今日哈工大的网站之后，本地中立刻出现了cache文件，同时在我们的程序窗口中也显示了文件名以及缓存成功的提示。

名称	修改日期	类型	大小
x64	2023/10/19 22:14	文件夹	
FileName.cpp	2023/10/20 0:36	C++ 源文件	18 KB
httptodayhiteducn.txt	2023/10/20 2:04	文本文档	65 KB
httptodayhiteducnthemescustomhit...	2023/10/20 2:04	文本文档	1 KB
LAB_1.sln	2023/10/19 22:13	Visual Studio Sol...	2 KB
LAB_1.vcxproj	2023/10/19 22:14	VCXPROJ 文件	7 KB
LAB_1.vcxproj.filters	2023/10/19 22:14	VC++ Project Fil...	1 KB
LAB_1.vcxproj.user	2023/10/19 22:13	Per-User Project...	1 KB

```

代理连接主机 today.hit.edu.cn 成功
http://today.hit.edu.cn/
代理连接主机 today.hit.edu.cn 成功
filename : httptodayhiteducn.txt

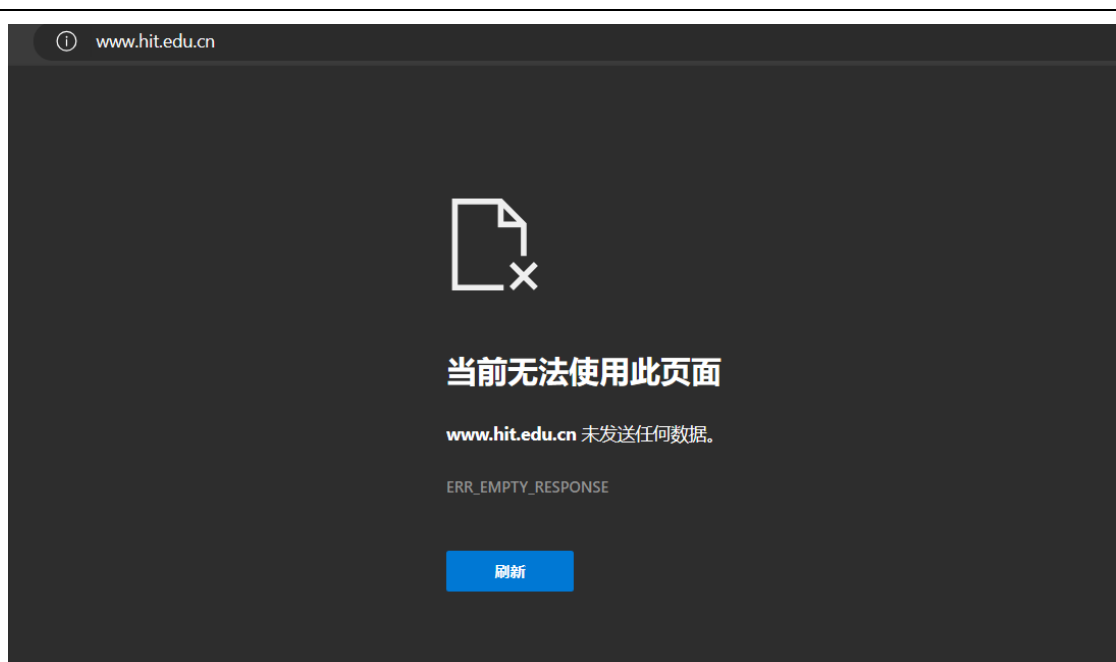
=====

*****网页已经被缓存*****
关闭套接字
    
```

当我们再次访问 <http://www.today.hit.edu.cn/> 网站时，我们加载的速度会比第一次快一点，而且在窗口中也显示，我们本次是通过获取本都缓存来加载的页面。

3. 扩展 http 代理服务器

(a) 网页过滤



```

关闭套接字
代理连接主机 www.hit.edu.cn 成功
*****不允许访问 http://www.hit.edu.cn/ *****
关闭套接字
    
```

我们将 `http://www.hit.edu.cn/` 作为过滤的网站，当我们打开程序，访问这个网址时，会提示无法运作，在我们的控制台中也显示提示信息禁止访问。说明成功实现网站过滤。

(b) 用户过滤

```

初始化...
代理服务器正在运行，监听端口 10240
获取用户IP地址：127.0.0.1
*****此IP已被禁用*****
    
```

我们把本机代理服务器的 IP 地址设为禁用，即禁用 `127.0.0.1`，然后运行程序，发现当识别到与禁用地址相同时，会退出程序，成功实现了用户禁用。

(c) 网站引导



```
代理连接主机 pku.edu.cn 成功
http://pku.edu.cn/
关闭套接字
关闭套接字
关闭套接字
关闭套接字
关闭套接字

*****读取本地缓存*****
date_str:Thu, 19 Oct 2023 18:04:08 GMT
代理连接主机 today.hit.edu.cn 成功

=====

*****从本机获得缓存*****
关闭套接字
关闭套接字
关闭套接字
关闭套接字
关闭套接字
代理连接主机 today.hit.edu.cn 成功
http://today.hit.edu.cn/themes/custom/hit_front/images/plus-colored.png
代理连接主机 today.hit.edu.cn 成功
```

我们一开始设置的网站引导是访问http://pku.edu.cn/重定向到 http://today.hit.edu.cn/, 然后启动程序, 访问pku.edu.cn, 经过处理后直接赚到了今日哈工大的页面, 至此, 网站引导功能完成。

问题讨论:

(1) 对于指导书中的参考代码报错的一些思考: a) 报错的第一个点就是#include "stdafx.h" 这一行代码报错, 解决方式就是直接去掉即可。错误的原因是没有这个头文件, 而这个头文件的作用经查询资料可知, Windows和MFC的include文件都非常大, 即使有一个快速的处理程序, 编译程序也要花费相当长的时间来完成工作。由于每个.CPP文件都包含相同的include文件, 为每个.CPP文件都重复处理这些文件耗时很长, 因此stdafx.h可以视为一个包含所有需要include的头文件, 在编译开始的时候先把这些内容编译好, 之后在编译每一个cpp文件的时候就阅读编译好的结果即可。b) 报错的第二个比较大的点就是代码中使用了goto, 而且在调用goto语句之后还定义了随机变量, 大部分编译器都会认为这种行为是危险的, 而在VS2019中可以通过对其进行简单设置去除这种错误, 经查阅资

料，其余一些编译器需要将所有的随机变量都定义在调用goto之前（事实上这样才是真正安全的，VS2019的处理方式只是强制略过这种错误检测）。

（2）对于使用代理服务器访问一些网站的时候存在图片无法加载或者加载的很慢的情况，猜测是因为对于代理服务器来说，HTTP发送报文的时候是不采用流水的，因此中间需要多次发送和接受过程，而且还经过了代理服务器这一中介，可能使得速度更慢，因此在现在的HTTP报文传送中猜测大多使用了流水线形式或其他形式加速发送报文。

心得体会：

本次实验让我对于socket编程的基本函数和基本操作有了认识，同时也了解了HTTP代理服务器的基本原理，对于HTTP请求和响应过程有了更深刻的认识。同时通过一些杜宇代理服务器的扩展，对于网站引导和屏蔽的最简单的实现方式有了认识，对于socket编程产生了浓厚兴趣。