实验 1：HTTP 代理服务器的设计与实现

1. 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。

2. 实验环境

- 接入 Internet 的实验主机；
- Windows 操作系统；
- 开发语言：C/C++（或 Java）等。

3. 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）

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）。

4. 实验步骤

（1）浏览器使用代理

为了使浏览器访问网址时通过代理服务器，必须进行相关设置，以 IE 浏览器设置为例：打开浏览器→工具→浏览器选项→连接→局域网设置→代理服务器，具体过程如图 1-1 所示。

图 1-1 浏览器的代理服务器设置

（2）多线程使用

使用函数_beginthreadex 创建子线程，使用函数_endthreadex 结束线程，详情见 CSDN。

（3）开发 HTTP 代理服务器并测试验证

编写 HTTP 代理服务器代码，部署运行，然后可以自己测试验证相关功能，也可以两位同学交叉验证。

5. 实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

6. 参考内容

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。如图 1-2 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。

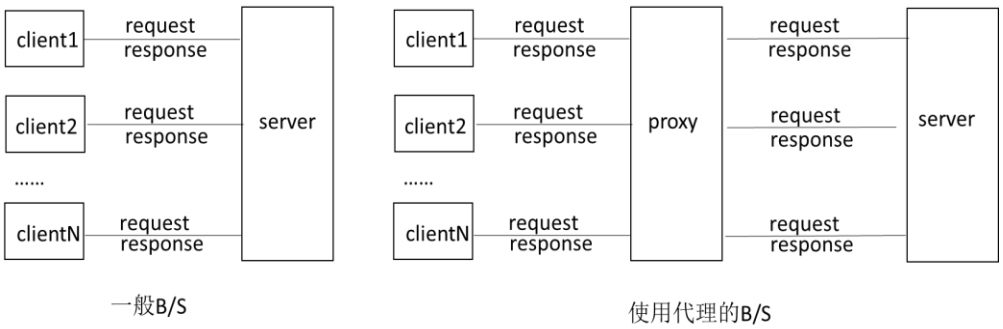

图 1-2 Web 应用通信方式对比

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。

本实验需实现一个简单的 HTTP 代理服务器，可以分为两个步骤：（首先请设置浏览器开启本地代理，注意设置代理端口与代理服务器监听端口保持一致）。

a) 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。

b) 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。

参考代码

```
#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.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));
    }
};

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;
sockaddr_in ProxyServerAddr;
const int ProxyPort = 10240;

//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;
```

```
};

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;
    ProxyParam *lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;
    //代理服务器不断监听
    while(true){
        acceptSocket = accept(ProxyServer,NULL,NULL);
        lpProxyParam = new ProxyParam;
        if(lpProxyParam == NULL){
            continue;
        }
        lpProxyParam->clientSocket = acceptSocket;
        hThread = (HANDLE)_beginthreadex(NULL, 0,
&ProxyThread,(LPVOID)lpProxyParam, 0, 0);
        CloseHandle(hThread);
        Sleep(200);
    }
    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;
    }
    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;
    ProxyServerAddr.sin_port = htons(ProxyPort);
```

```
        ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
        if(bind(ProxyServer,(SOCKADDR*)&ProxyServerAddr,sizeof(SOCKADD
R)) == SOCKET_ERROR){
            printf("绑定套接字失败\n");
            return FALSE;
        }
        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;
        ZeroMemory(Buffer,MAXSIZE);
        SOCKADDR_IN clientAddr;
        int length = sizeof(SOCKADDR_IN);
        int recvSize;
        int ret;
        recvSize = recv(((ProxyParam
*)lpParameter)->clientSocket,Buffer,MAXSIZE,0);
        if(recvSize <= 0){
            goto error;
        }
        HttpHeader* httpHeader = new HttpHeader();
```

```
            CacheBuffer = new char[recvSize + 1];
            ZeroMemory(CacheBuffer,recvSize + 1);
            memcpy(CacheBuffer,Buffer,recvSize);
            ParseHttpHead(CacheBuffer,httpHeader);
            delete CacheBuffer;
            if(!ConnectToServer(&((ProxyParam
*)lpParameter)->serverSocket,httpHeader->host)) {
                  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;
            }
            //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");
            Sleep(200);
            closesocket(((ProxyParam*)lpParameter)->clientSocket);
            closesocket(((ProxyParam*)lpParameter)->serverSocket);
            delete    lpParameter;
            _endthreadex(0);
            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;
    }
```

7. 实验报告

在实验报告中需要总结说明：

（1）Socket 编程的客户端和服务器端主要步骤；

（2）HTTP 代理服务器的基本原理；

（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；

（5）HTTP 代理服务器实验验证过程以及实验结果；

（6）HTTP 代理服务器源代码（带有详细注释）。