哈爾濱工業大學
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	李旻翀		院系	计算机科学与技术	
班级	1903103		学号	1190200208	
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	2021.10.31	
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					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

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 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）

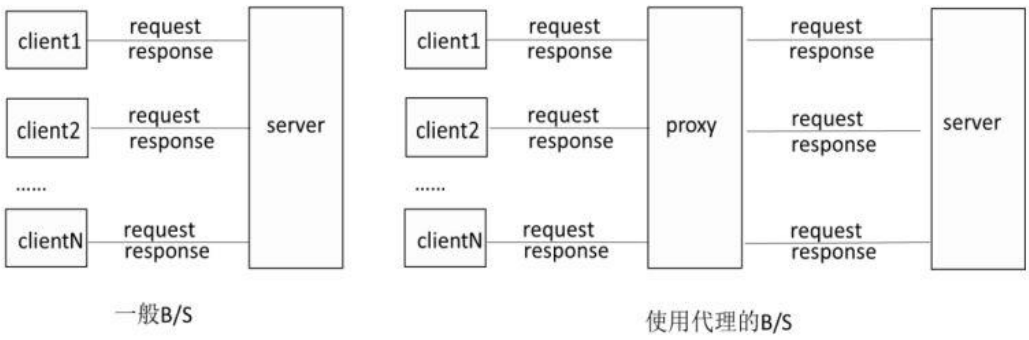
a) 网站过滤：允许/不允许访问某些网站；
b) 用户过滤：支持/不支持某些用户访问外部网站；
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

实验过程：

1. 了解实验相关基础知识

1) 代理服务器的概念

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。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如下图所示：



代理服务器可以认为是TCP/IP网络应用的客户端和服务器端的结合。一方面，它是浏览器客户端的服务器端，另一方面，它也是目标服务器的客户端。浏览器将请求报文发送给代理服务器，代理服务器经过一些处理或者不经过处理，将请求报文转发给目标服务器；目标服务器相应请求报文发出响应报文，代理服务器接受到响应报文之后直接将响应报文转发给浏览器客户端。

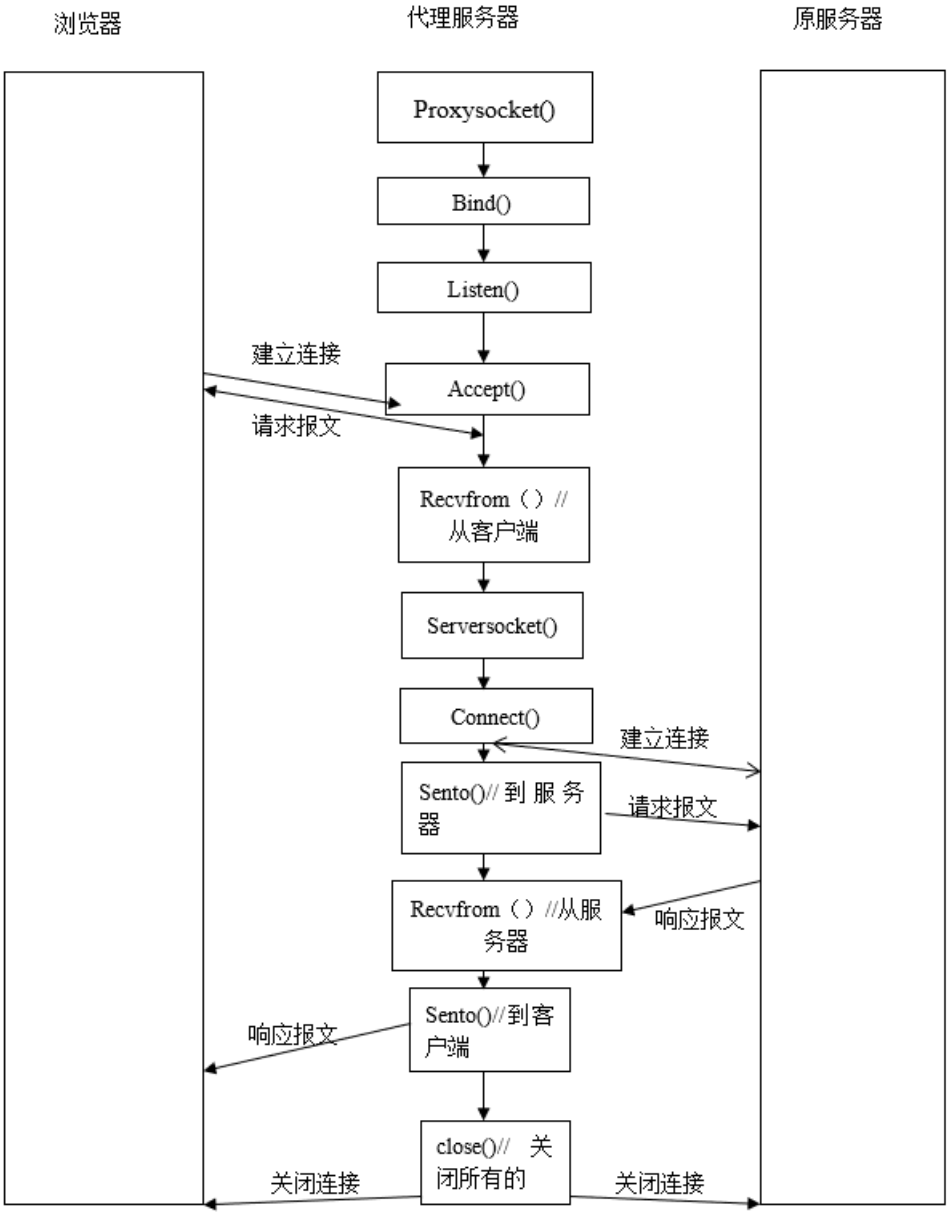
针对本实验的特定要求，概括我们所要实现的代理服务器功能如下：

代理服务器在指定端口（本实验中所指定的是10240端口）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），代理服务器接收到浏览器对远程网站的浏览请求时，首先会查看浏览器来源的IP地址，如果属于被限制的用户，则认为没有接受到访问请求（用户过滤功能）。否则，查看其请求的host主机，如果属于不允许访问的主机，则默认不向目标服务器发送请求（网站过滤功能）；如果属于被引导的网站，则对该网站的请求报文中的host主机

地址和url进行更改（网站引导功能）。而对于Cache功能的实现，基本可以概括为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），若找到对象文件，则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。若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。

除此之外，本实验要设计的服务器属于多用户代理服务器。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。

HTTP代理服务器的流程图如下：

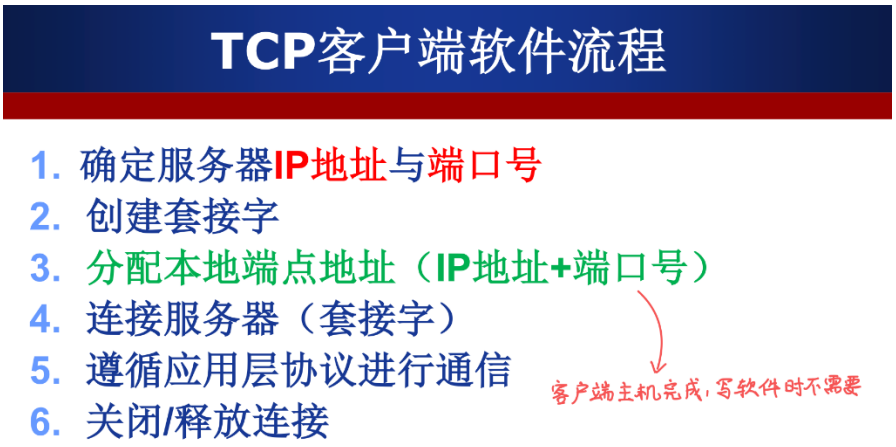


2) TCP客户端与软件端的流程

(a) TCP客户端软件流程

1. 根据目标服务器IP地址与端口号创建套接字（socket），
2. 连接服务器（connect）：三次握手
3. 发送请求报文（send）
4. 接收返回报文（recv），返回3或者5
5. 关闭连接（closesocket）

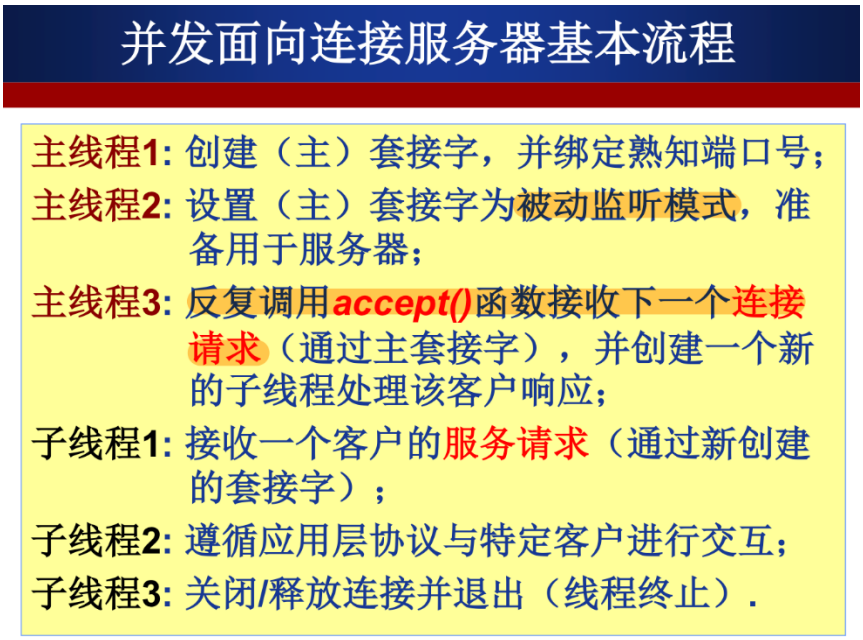
根据上课的PPT，可以总结为：



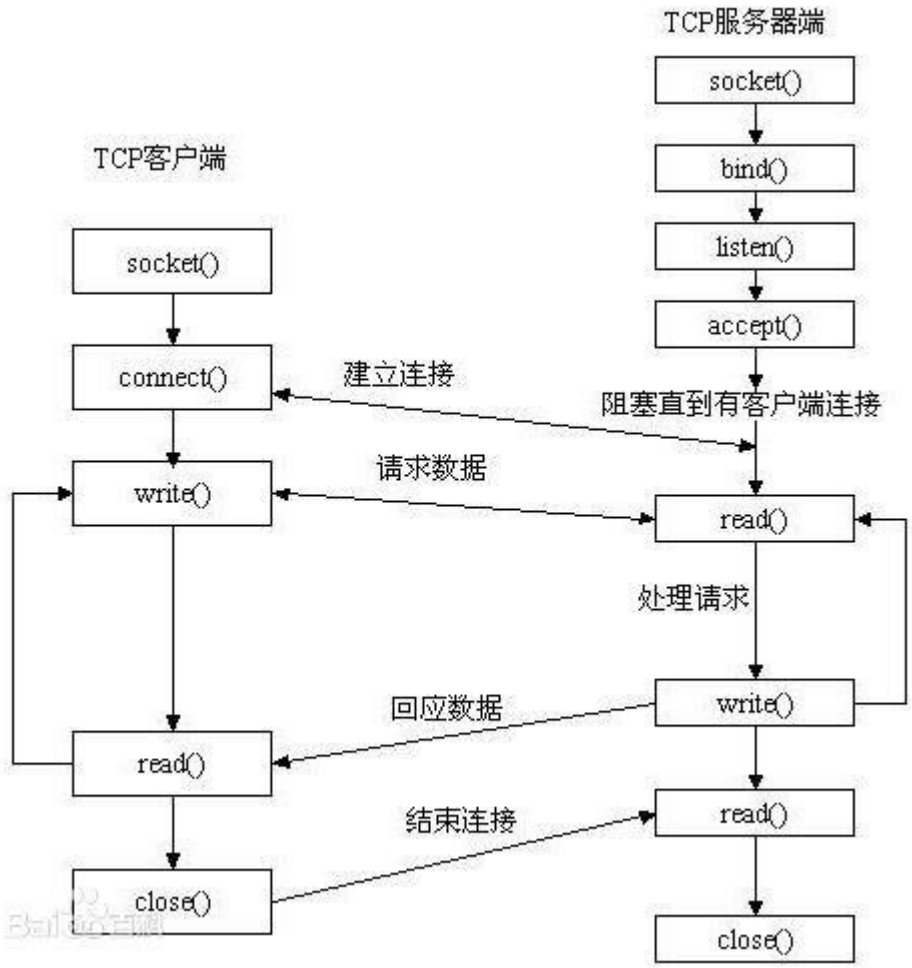
(b) TCP服务器端软件流程

1. 创建套接字（socket），绑定套接字的本地IP地址和端口号（bind），然后转到监听模式并设置连接请求队列大小（listen）。
2. 从连接请求队列中取出一个连接请求，并同意连接（accept）。在TCP连接过程中进行了三次握手。
3. 收到请求报文（recv）
4. 发送数据（send）返回3或者5
5. 关闭连接（closesocket）返回2

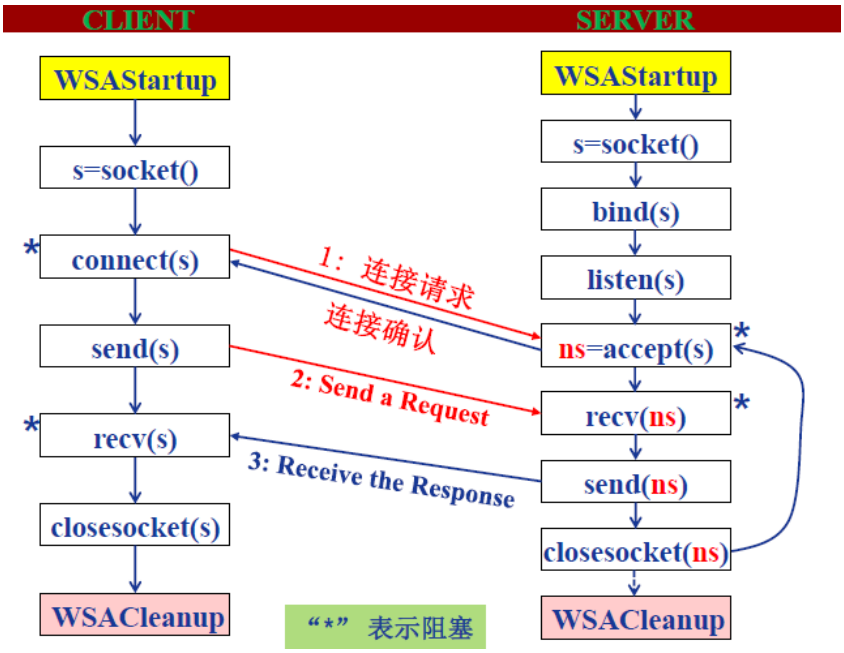
根据上课的PPT，可以总结为：



(c) TCP软件端与服务器端流程图

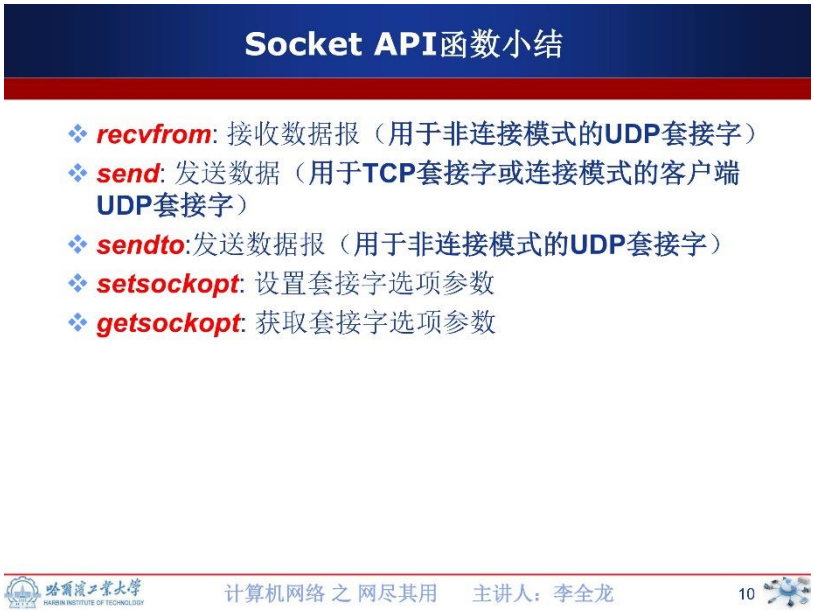


3) 网络应用的Socket API(TCP)调用基本流程



4) 常用C Socket编程函数及其功能

根据MOOC上PPT内容总结如下：



2. 修改程序，完成基本功能

对于代理服务器的实现，我采用基于C语言示例代码修改的方式完成。由于我使用的编程环境为VS Code而非VS，程序初始提供的示例代码部分地方无法使用，因此，经过查询资料，发现有以下地方需要修改：

1) 主函数部分修改

由于编程环境问题，不支持 int _tmain(int argc, _TCHAR* argv[]) 的写法，因此，可以将其修改为 int main() 或 int main (int argc, char* argv[])。

2) goto语句后不能再定义变量

在运行示例代码时，会出现如图所示的错误：

```
main.cpp: In function 'unsigned int ProxyThread(LPVOID)':
main.cpp:358:1: error: jump to label 'error' [-fpermissive]
 error:
 ^~~~~
main.cpp:220:8: note:   from here
   goto error;
        ^~~~~
```

经过查阅相关资料，发现是在 goto 语句之后不能再定义新的变量，因此将相关代码注释即可：

```
// 为避免报jump to label 'error'的错误，将其注释
// if (recvSize <= 0)
// {
//  goto error;
// }
```

3) 静态链接 Ws2_32.lib 的问题

示例代码修改无误后仍然无法运行，会出现很多引用无效的错误提示：

```
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xbb): undefined reference to `__imp_accept'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xcd): undefined reference to `__imp_inet_ntoa'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x108): undefined reference to `__imp_closesocket'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x1bc): undefined reference to `__imp_WSAStartup'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x1d4): undefined reference to `__imp_WSAGetLastError'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x215): undefined reference to `__imp_WSACleanup'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x238): undefined reference to `__imp_socket'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x255): undefined reference to `__imp_WSAGetLastError'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x284): undefined reference to `__imp_htons'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x2ab): undefined reference to `__imp_bind'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x2e0): undefined reference to `__imp_listen'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x3b0): undefined reference to `__imp_recv'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x7d6): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x917): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x944): undefined reference to `__imp_recv'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xa73): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xb84): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xbe3): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xc13): undefined reference to `__imp_recv'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xc4b): undefined reference to `__imp_send'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xca7): undefined reference to `__imp_closesocket'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0xcbe): undefined reference to `__imp_closesocket'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x155c): undefined reference to `__imp_htons'
C:\Users\LMC117\AppData\Local\Temp\ccjpYaaP.o:main.cpp:(.text+0x1570): undefined reference to `__imp_gethostbyname'
```

查阅相关资料，发现是这行代码的问题：

```
#pragma comment(lib, "Ws2_32.lib")
```

因为#pragma是VS的写法，而VS Code内的C编译器MingGW不支持该表示格式，因此，想要运行该程序，我们需要使用g++对Ws2_32.lib进行静态链接，相关语句如下：

```
(base) PS C:\Users\LMC117\Desktop\计网参考\Lab1> g++ main.cpp -l ws2_32 -o main.exe
```

运行后，可以在相同文件夹下得到main.exe文件，执行该文件，即可得到我们代码的运行结果。

3. 程序基本功能描述

在修改完成后，示例代码便可正常运行，以下简述示例代码的流程与功能：

1) 初始化一个套接字，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”；同时也要绑定端口号，在示例代码中，端口号被设置为“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；同时，服务器也将获得的响应发给代理服务器，然后代理服务器再将该响应发送给客户端。在这里，代理服务器相当于一个中介，提供一个代理的服务，所有的请求和响应都经过它。
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。对于客户端而言，它只要将正常发送的请求发给代理服务器，就可以接收到对应的响应。

4. 拓展HTTP代理服务器（网站过滤，用户过滤，网站引导）

为便于后续实验验证，进行如下设定。

定义如下宏常量：

```
#define BANNED_WEB "http://today.hit.edu.cn/"        //屏蔽网站
#define PHISHING_WEB_SRC "http://jwc.hit.edu.cn/"    // 钓鱼原网址
#define PHISHING_WEB_DEST "http://jwts.hit.edu.cn/" // 钓鱼目的网址
```

设置被屏蔽的网站为今日哈工大网站（http://today.hit.edu.cn/），设置钓鱼网站为由哈工大教务处（http://jwc.hit.edu.cn/）重定向至选课系统（http://jwts.hit.edu.cn/），相关示例网站可以更换，但需要注意，更换的示例网站必须采用HTTP协议，而非HTTPS协议。

1) 网站过滤

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。

```
// 网站屏蔽
if (strcmp(httpHeader->url, BANNED_WEB) == 0)
{
    printf("网站 %s 已被屏蔽\n", BANNED_WEB);
    goto error;
}
```

2) 用户过滤

修改ProxyServerAddr.sin_addr.S_un.S_addr的值为inet_addr("127.0.0.1")，表示只允许本机访问代理服务器，从而实现用户过滤功能。代码如下：

```
// ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //只允许本机用户访问服务器
```

3) 网站引导

网站引导功能同样在线程执行函数ProxyThread中实现。实现思路为：比较HTTP头部中的URL字段是否与PHISHING_WEB_SRC相同，若相同，则说明客户端想要访问的网址是重定向源网址，需要执行重定向操作。具体而言，在重定向操作中，我们分部分构造302报文，将其中的IP地址修改为钓鱼目的IP地址，在最后，将修改好的302报文通过send函数返回给客户端。

网站引导的核心函数如下图所示：

```
//网站钓鱼  访问jwc.hit.edu.cn  重定向到jwts.hit.edu.cn
if (strstr(httpHeader->url, PHISHING_WEB_SRC) != NULL)
{
    char *pr;
    int phishing_len;
    // 打印信息
    printf("网站 %s 已被成功重定向至 %s\n", PHISHING_WEB_SRC, PHISHING_WEB_DEST);
    // 构造报文
    char head1[] = "HTTP/1.1 302 Moved Temporarily\r\n";
    phishing_len = strlen(head1);
    memcpy(phishBuffer, head1, phishing_len);
    pr = phishBuffer + phishing_len;

    char head2[] = "Connection:keep-alive\r\n";
    phishing_len = strlen(head2);
    memcpy(pr, head2, phishing_len);
    pr += phishing_len;

    char head3[] = "Cache-Control:max-age=0\r\n";
    phishing_len = strlen(head3);
    memcpy(pr, head3, phishing_len);
    pr += phishing_len;

    //重定向到jwts.hit.edu.cn
    char phishing_dest[] = "Location: ";
    strcat(phishing_dest, PHISHING_WEB_DEST);
    strcat(phishing_dest, "\r\n\r\n");
    phishing_len = strlen(phishing_dest);
    memcpy(pr, phishing_dest, phishing_len);

    //将302报文返回给客户端
    ret = send(((ProxyParam *)lpParameter)->clientSocket, phishBuffer, sizeof(
phishBuffer), 0);
    goto error;
}
```

5. 实现Cache功能

在本次代理服务器的实验中，Cache的基本功能如下：

代理服务器第一次和客户端通信时，会保留Cache；当客户端再次请求本地存在的 cache 页面时，代理服务器会通过 If-Modified-Since 头将先前目标服务器端发过来的 Last-Modified 最后

修改时间戳发送回去，让目标服务器端进行验证，通过这个时间戳判断客户端的页面是否是最新的，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。

具体到代码实现，首先，通过我们修改后的ParseHttpHead函数解析TCP报文中的HTTP头部，并在解析的同时判断HTTP头中包含的url是否已经存在于Cache中，若是，则会将该url存入代理服务器的Cache中。如果代理服务器的Cache没有了空间，则覆盖Cache的第一个位置。ParseHttpHead函数的返回值Have_cache标志着请求的页面在代理服务器上是否有缓存，若是，则构造缓存的报文头，由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>，然后将客户端发送的HTTP数据报文直接转发给目标服务器，并等待目标服务器返回数据。在目标服务器返回数据后，由代理服务器解析包含缓存信息的HTTP报文头，读取返回的状态字与页面的最后修改时间。若状态码为304，则说明页面未被修改，打印相关信息后直接由代理服务器将缓存数据转发给客户端；若状态码为200，则表示文件已被修改，首先修改缓存内容，然后再将目标服务器返回的数据直接转发给客户端。若是代理服务器根本没有缓存过该页面，则将该页面缓存到Cache中，然后将客户端发送的HTTP数据报文直接转发给目标服务器，再将目标服务器返回的数据直接转发给客户端。在此过程中，任何一步出现了错误，都直接通过goto语句跳转到error进行关闭套接字，结束该线程的处理。

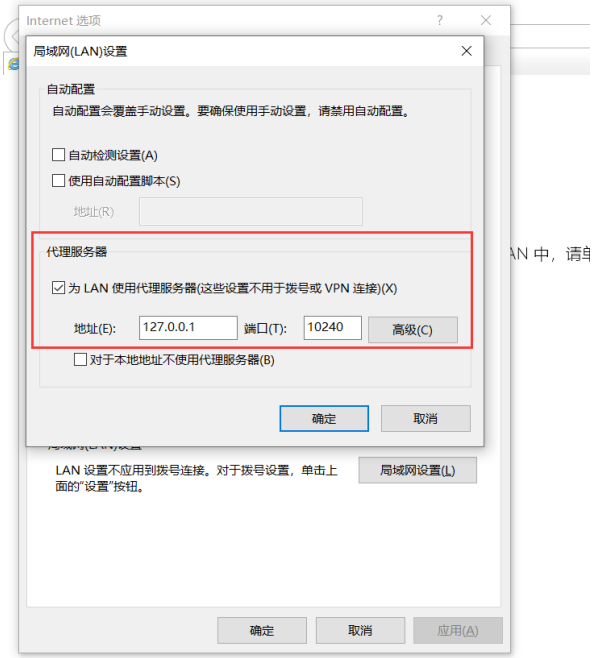
由于相关源代码较多，无法放在一页中，故此处暂不对该部分代码做展示。

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

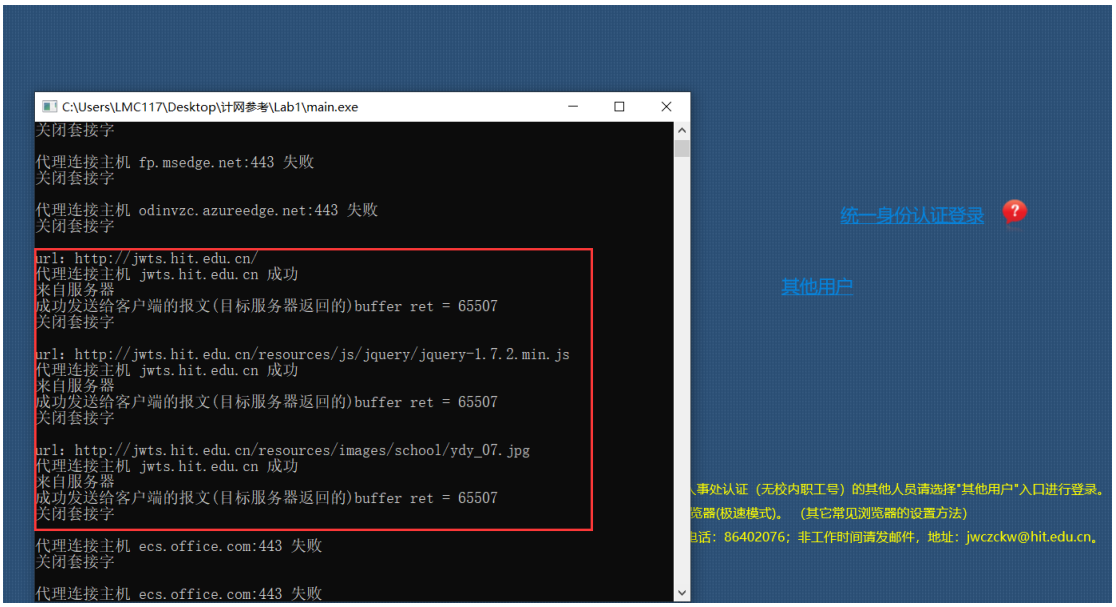
1. 修改IE浏览器的代理设置

按照实验指导书修改IE浏览器设置如下：



2. 通过代码实现静态链接，并运行程序，实现基础功能

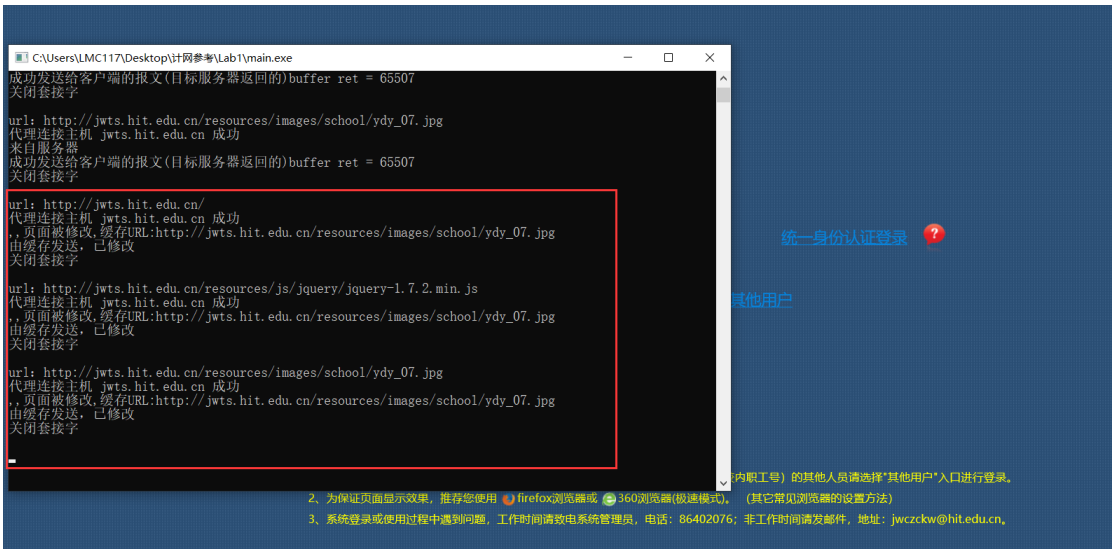
输入g++ main.cpp -l ws2_32 -o main.exe生成可执行文件main.exe，点击运行，然后打开IE浏览器，打开一个使用http协议的网站，如：http://jwts.hit.edu.cn/，可以看到如下结果：



这说明我们的代理服务器成功接收了来自客户的 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客户进行浏览。

3. 验证代理服务器的Cache功能

再次在IE浏览器地址栏输入http://jwts.hit.edu.cn/进行访问，结果如下：



4. 验证代理服务器的HTTP拓展功能

1) 网站过滤

设定的屏蔽网站为http://today.hit.edu.cn，输入网址尝试进行访问：

2) 用户过滤

修改程序代码，将允许访问的用户IP地址修改为127.0.0.2，如图所示：

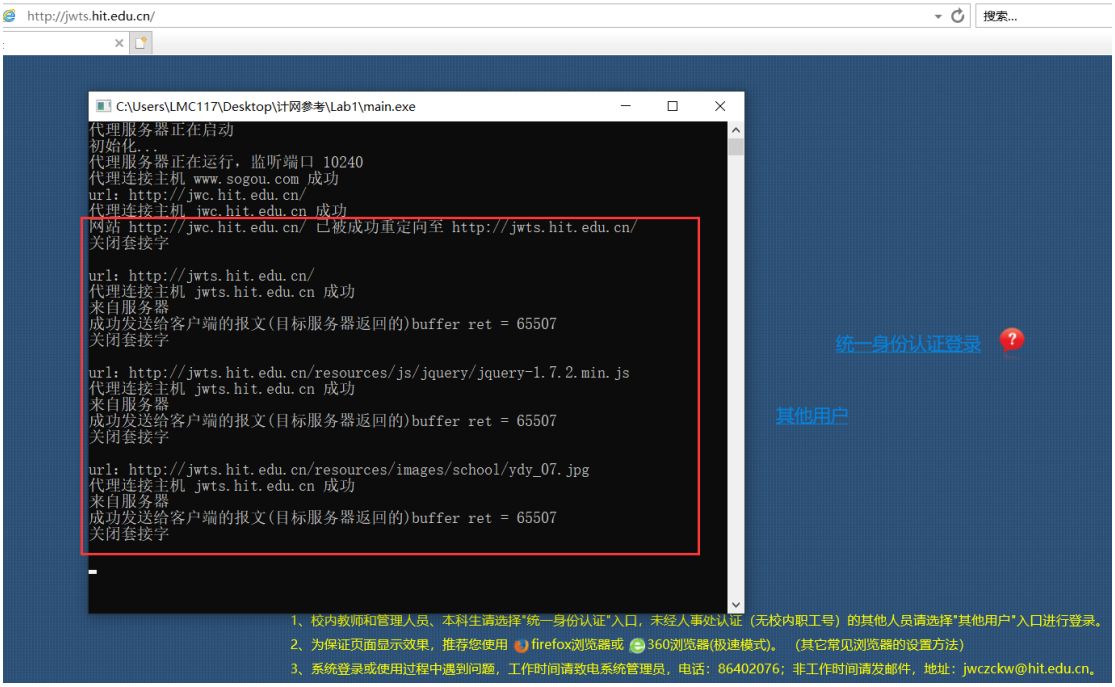
```
ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.2"); //只允许本机用户访问服务器
```

重新生成程序进行运行：

可以看出，代理服务器不会对我们的访问做出相应，X用户被成功屏蔽。

3) 网站引导

设定的钓鱼源网站为http://jwc.hit.edu.cn，目的网站为http://jwts.hit.edu.cn，在地址栏输入http://jwc.hit.edu.cn尝试进行访问：



网页被成功重定向。

心得体会：

结合实验过程和结果给出实验的体会和收获。

- 对TCP协议传输数据的流程和方式有了更深的体会；
- 对 socket 编程方法有了初步的了解，通过动手实践有了很大收获；
- 了解了 HTTP 代理服务器的基本原理，掌握了 HTTP 代理服务器设计与编程实现的基本技能，对 HTTP 请求和响应原理有了更深的认识；
- 对网站钓鱼、网站屏蔽、用户屏蔽的机制有了深刻的理解；
- 对 HTTP Cache缓存的作用有了直观印象

实验源码：

```
//#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>
#include <tchar.h>
#include <fstream>
#include <map>
#include <string>

#include <iostream>
using namespace std;

#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

#define BANNED_WEB "http://today.hit.edu.cn/"       //屏蔽网站
#define PHISHING_WEB_SRC "http://jwc.hit.edu.cn/"   // 钓鱼原网址
#define PHISHING_WEB_DEST "http://jwts.hit.edu.cn/" // 钓鱼目的网址

//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));
    }
};

// 结构体 cache
map<string, char *> cache;
struct HttpCache
{
    char url[1024];
    char host[1024];
    char last_modified[200];
    char status[4];
```

```
    char buffer[MAXSIZE];
    HttpCache()
    {
        ZeroMemory(this, sizeof(HttpCache)); // 初始化 cache
    }
};
HttpCache Cache[1024];
int cached_number = 0; //已经缓存的url 数
int last_cache = 0;    //上一次缓存的索引

BOOL InitSocket();
int ParseHttpHead(char *buffer, HttpHeader *httpHeader);
BOOL ConnectToServer(SOCKET *serverSocket, char *host);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
void ParseCache(char *buffer, char *status, char *last_modified);

//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 main(int argc, char *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;
```

```
    SOCKADDR_IN acceptAddr;
    ProxyParam *lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;

    //代理服务器不断监听
    while (true)
    {
        acceptSocket = accept(ProxyServer, (SOCKADDR *)&acceptAddr, 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 Socket 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); // 创建一个 TCP/IP 协议族的
流套接字
    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;
    ProxyServerAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1"); //只允许本
机用户访问服务器

    if (bind(ProxyServer, (SOCKADDR *)&ProxyServerAddr, sizeof(SOCKADDR)) ==
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 sendBuffer[MAXSIZE];
    char phishBuffer[MAXSIZE];
    char *CacheBuffer;

    ZeroMemory(Buffer, MAXSIZE);
    ZeroMemory(sendBuffer, MAXSIZE);
    ZeroMemory(phishBuffer, MAXSIZE);

    SOCKADDR_IN clientAddr;
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    int Have_cache;

    //接收客户端的请求
    recvSize = recv(((ProxyParam *)lpParameter)->clientSocket, Buffer,
MAXSIZE, 0);

    // 为避免报 jump to label 'error' 的错误，将其注释
    // if (recvSize <= 0)
    // {
    //  goto error;
    // }

    HttpHeader *httpHeader = new HttpHeader();
    memcpy(sendBuffer, Buffer, recvSize);
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer, recvSize + 1);
    memcpy(CacheBuffer, Buffer, recvSize);
    //ParseHttpHead(CacheBuffer, httpHeader);
    Have_cache = ParseHttpHead(CacheBuffer, httpHeader);
    delete CacheBuffer;

    if (!ConnectToServer(&((ProxyParam *)lpParameter)->serverSocket,
httpHeader->host))
    {
        printf("代理连接主机 %s 失败\n", httpHeader->host);
```

```
        goto error;
    }
    printf("代理连接主机 %s 成功\n", httpHeader->host);

    // 网站屏蔽
    if (strcmp(httpHeader->url, BANNED_WEB) == 0)
    {
        printf("网站 %s 已被屏蔽\n", BANNED_WEB);
        goto error;
    }

    //网站钓鱼  访问jwc.hit.edu.cn  重定向到jwts.hit.edu.cn
    if (strstr(httpHeader->url, PHISHING_WEB_SRC) != NULL)
    {
        char *pr;
        int phishing_len;
        // 打印信息
        printf("网站 %s 已被成功重定向至 %s\n", PHISHING_WEB_SRC,
PHISHING_WEB_DEST);
        // 构造报文
        char head1[] = "HTTP/1.1 302 Moved Temporarily\r\n";
        phishing_len = strlen(head1);
        memcpy(phishBuffer, head1, phishing_len);
        pr = phishBuffer + phishing_len;

        char head2[] = "Connection:keep-alive\r\n";
        phishing_len = strlen(head2);
        memcpy(pr, head2, phishing_len);
        pr += phishing_len;

        char head3[] = "Cache-Control:max-age=0\r\n";
        phishing_len = strlen(head3);
        memcpy(pr, head3, phishing_len);
        pr += phishing_len;

        //重定向到jwts.hit.edu.cn
        char phishing_dest[] = "Location: ";
        strcat(phishing_dest, PHISHING_WEB_DEST);
        strcat(phishing_dest, "\r\n\r\n");
        phishing_len = strlen(phishing_dest);
        memcpy(pr, phishing_dest, phishing_len);

        //将302报文返回给客户端
```

```
        ret = send(((ProxyParam *)lpParameter)->clientSocket, phishBuffer,
sizeof(phishBuffer), 0);
        goto error;
    }

    //实现 cache 功能
    if (Have_cache) //请求的页面在服务器有缓存
    {
        char cached_buffer[MAXSIZE];
        ZeroMemory(cached_buffer, MAXSIZE);
        memcpy(cached_buffer, Buffer, recvSize);

        //构造缓存的报文头
        char *pr = cached_buffer + recvSize;
        printf(",,");
        memcpy(pr, "If-modified-since: ", 19);
        pr += 19;
        int length = strlen(Cache[last_cache].last_modified);
        memcpy(pr, Cache[last_cache].last_modified, length);
        pr += length;

        //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, cached_buffer,
strlen(cached_buffer) + 1, 0);
        //等待目标服务器返回数据
        recvSize = recv(((ProxyParam *)lpParameter)->serverSocket,
cached_buffer, MAXSIZE, 0);
        if (recvSize <= 0)
        {
            goto error;
        }

        //解析包含缓存信息的HTTP 报文头
        CacheBuffer = new char[recvSize + 1];
        ZeroMemory(CacheBuffer, recvSize + 1);
        memcpy(CacheBuffer, cached_buffer, recvSize);

        char last_status[4];    //记录主机返回的状态字
        char last_modified[30]; //记录返回页面的修改时间
        ParseCache(CacheBuffer, last_status, last_modified);

        delete CacheBuffer;

        //分析 cache 的状态字
```

```
        if (strcmp(last_status, "304") == 0) //304 状态码，文件没有被修改
        {
            printf("页面未被修改,缓存 URL:%s\n", Cache[last_cache].url);
            //直接将缓存数据转发给客户端
            ret = send(((ProxyParam *)lpParameter)->clientSocket,
Cache[last_cache].buffer, sizeof(Cache[last_cache].buffer), 0);
            if (ret != SOCKET_ERROR)
                printf("由缓存发送\n");
        }
        else if (strcmp(last_status, "200") == 0) //200 状态码，表示文件已被修
改
        {
            //首先修改缓存内容
            printf("页面被修改,缓存 URL:%s\n", Cache[last_cache].url);
            memcpy(Cache[last_cache].buffer, cached_buffer,
strlen(cached_buffer));
            memcpy(Cache[last_cache].last_modified, last_modified,
strlen(last_modified));

            //将目标服务器返回的数据直接转发给客户端
            ret = send(((ProxyParam *)lpParameter)->clientSocket,
cached_buffer, sizeof(cached_buffer), 0);
            if (ret != SOCKET_ERROR)
                printf("由缓存发送，已修改\n");
        }
    }
    else //没有缓存过该页面
    {
        //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;
        }

        //将该页面缓存到 cache 中

        //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);
```

```
        if (ret != SOCKET_ERROR)
        {
            printf("来自服务器\n 成功发送给客户端的报文(目标服务器返回的)buffer
ret = %d \n", ret);
        }
    }
    //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);
    return 0;
}

//************************
//Method: ParseCache
//FullName: ParseCache
//Access: public
//Returns: void
//Qualifier: 解析 TCP 报文中的 HTTP 头部,在已经 cache 命中的时候使用
//Parameter: char *buffer
//Parameter: char * status
//Parameter: HttpHeader *httpHeader
//************************
void ParseCache(char *buffer, char *status, char *last_modified)
{
    char *p;
    char *ptr;
    const char *delim = "\r\n";
    p = strtok_s(buffer, delim, &ptr); //提取第一行
    memcpy(status, &p[9], 3);
    status[3] = '\0';
    p = strtok_s(NULL, delim, &ptr);
    while (p)
    {
        if (strstr(p, "Last-Modified") != NULL)
        {
            memcpy(last_modified, &p[15], strlen(p) - 15);
            break;
        }
        p = strtok_s(NULL, delim, &ptr);
```

```
    }
}

//************************
//Method: ParseHttpHead
//FullName: ParseHttpHead
//Access: public
//Returns: int
//Qualifier: 解析 TCP 报文中的 HTTP 头部
//Parameter: char *buffer
//Parameter: HttpHeader *httpHeader
//************************
int ParseHttpHead(char *buffer, HttpHeader *httpHeader)
{
    int flag = 0; //用于表示Cache 是否命中，命中为1，不命中为0
    char *p;
    char *ptr;
    const char *delim = "\r\n"; //回车换行符
    p = strtok_s(buffer, delim, &ptr);
    if (p[0] == 'G')
    { //GET 方式
        memcpy(httpHeader->method, "GET", 3);
        memcpy(httpHeader->url, &p[4], strlen(p) - 13);
        printf("url: %s\n", httpHeader->url); //url
        for (int i = 0; i < 1024; i++)
        { //搜索 cache，看当前访问的url 是否已经存在cache 中了
            if (strcmp(Cache[i].url, httpHeader->url) == 0)
            { //说明url 在cache 中已经存在
                flag = 1;
                break;
            }
        }
        if (!flag && cached_number != 1023) //说明url 没有在cache 且cache 没有
满，把这个url 直接存进去
        {
            memcpy(Cache[cached_number].url, &p[4], strlen(p) - 13);
            last_cache = cached_number;
        }
        else if (!flag && cached_number == 1023) //说明url 没有在cache 且
cache 满了,把第一个cache 覆盖
        {
            memcpy(Cache[0].url, &p[4], strlen(p) - 13);
            last_cache = 0;
        }
```

```
    }
    else if (p[0] == 'P') //POST 方式
    {
        memcpy(httpHeader->method, "POST", 4);
        memcpy(httpHeader->url, &p[5], strlen(p) - 14);
        for (int i = 0; i < 1024; i++)
        {
            if (strcmp(Cache[i].url, httpHeader->url) == 0)
            {
                flag = 1;
                break;
            }
        }
        if (!flag && cached_number != 1023)
        {
            memcpy(Cache[cached_number].url, &p[5], strlen(p) - 14);
            last_cache = cached_number;
        }
        else if (!flag && cached_number == 1023)
        {
            memcpy(Cache[0].url, &p[4], strlen(p) - 13);
            last_cache = 0;
        }
    }

    p = strtok_s(NULL, delim, &ptr);
    while (p)
    {
        switch (p[0])
        {
        case 'H': //HOST
            memcpy(httpHeader->host, &p[6], strlen(p) - 6);
            if (!flag && cached_number != 1023)
            {
                memcpy(Cache[last_cache].host, &p[6], strlen(p) - 6);
                cached_number++;
            }
            else if (!flag && cached_number == 1023)
            {
                memcpy(Cache[last_cache].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;
            //case '':
        default:
            break;
        }
        p = strtok_s(NULL, delim, &ptr);
    }
    return flag;
}

//************************************
// 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;
}
```