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	王炳轩		院系	计算学部-信息安全	
班级	2003201		学号	120L022115	
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	2022 年 10 月 7 日	
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：

本次实验的主要目的。

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：

概述本次实验的主要内容，包含的实验项等。

(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；

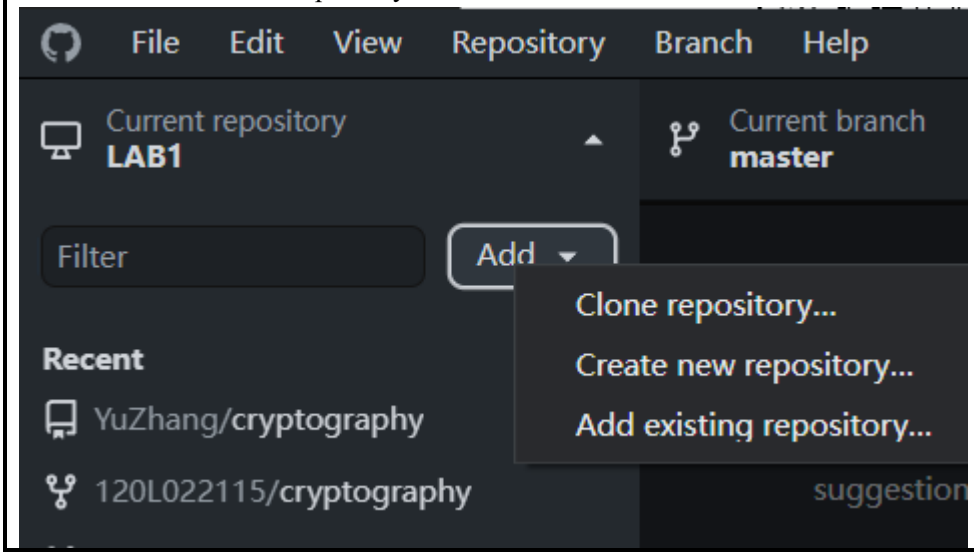
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

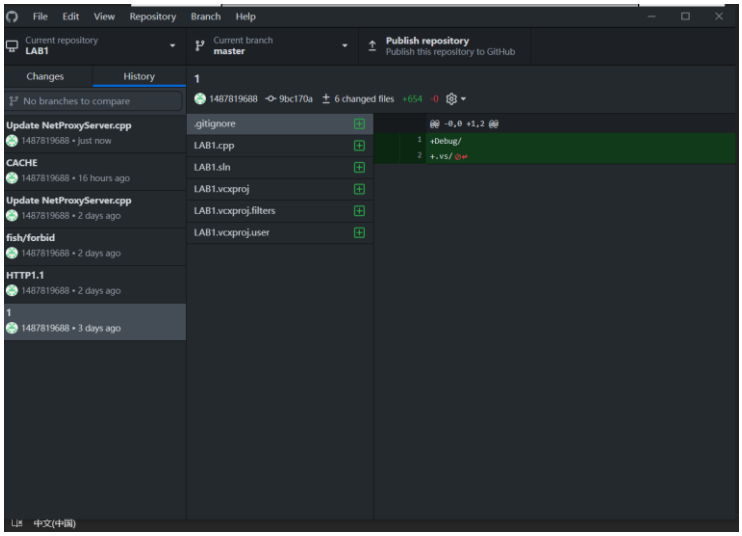
实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

一、首先创建Git本地仓库，用以保存实验文件的历史版本。

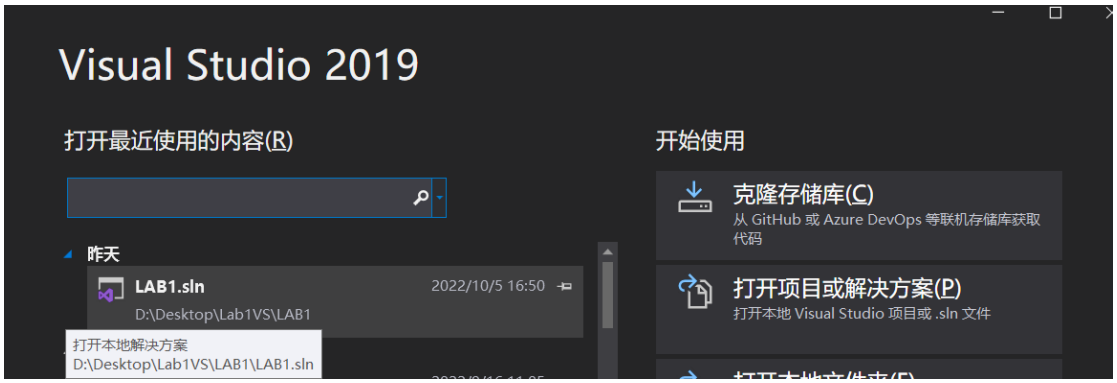
下载安装Github Desktop应用，点击当前仓库列表（Current Repository），点击新建（Add）-本地仓库（Create new repository）。输入仓库名为LAB1，点击确认创建。

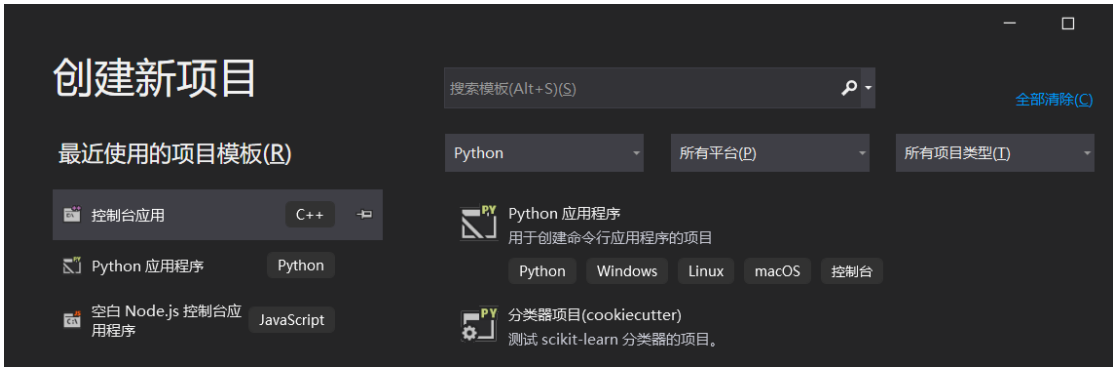


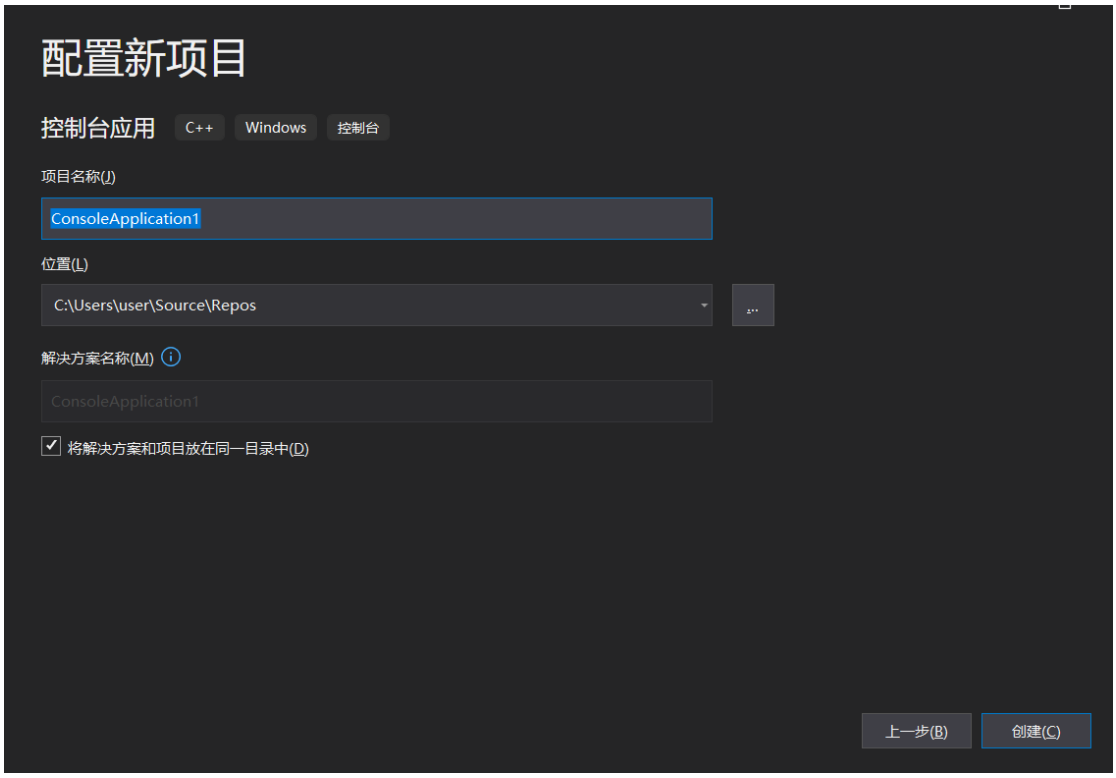


二、配置Visual Studios 2019，新建解决方案LAB1。

打开VS2019，选择创建新项目，选择C/C++控制台程序，输入项目名称和存储位置，点击创建。

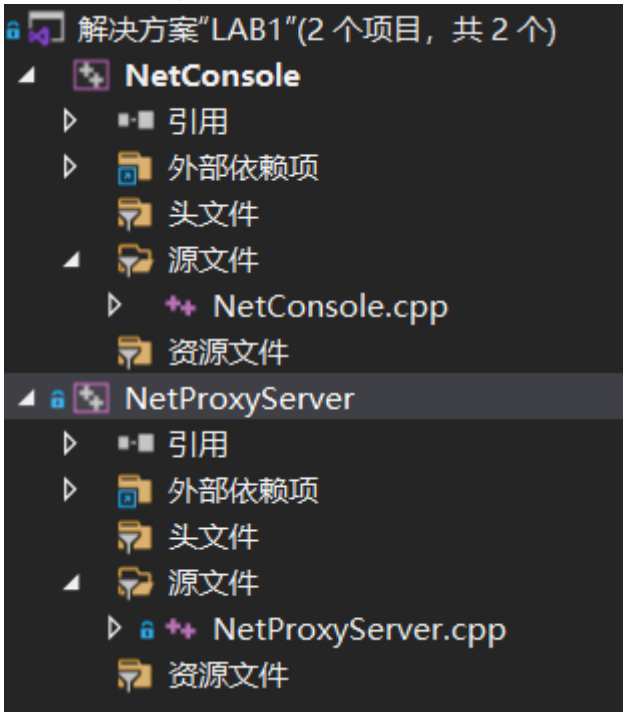






三、构建实验框架：服务器和控制台。

如图所示，在解决方案LAB1中，新建两个项目，分别为NetConsole和NetProxyServer，其中NetConsole用以控制和显示代理服务器的基本信息、连接情况以及控制代理服务器的行为等，而NetProxyServer则以存储-转发的形式在本机的10240端口上作为代理服务器进行工作。



四、编写实验代码

4.1 学习和修正示例代码以实现HTTP1.0

首先打开指导书，学习相关要求并复制示例代码到NetProxyServer.cpp中，并进行阅读和结构规划。

发现示例代码中含有多处错误，包含栈溢出、堆地址错误、线程不安全、内存泄漏、传参错误、HTTP报文未完整接收等。

对这些代码进行检查、修复：

1、首先对内存泄漏和堆地址错误进行修复，经过调试，发现是示例代码中new和delete非成对出现，存在new后未delete（造成内存泄漏）和双重delete的野指针（造成堆地址的非法访问和DLL被意外删除导致未加载）等情况，对以上内存申请进行修复，保证new和delete的成对性。

2、然后对栈溢出进行修复，将使用栈的内存转移到堆内存或静态内存区中，如将静态大数组char *Buffer[65507]声明更改为动态申请new char[65507]，并在后续代码中加入delete[]，并在项目配置中增大堆栈保留空间。

3、然后对线程不安全的函数进行修复，更改为使用推荐的线程安全函数如：fopen_s, strtok_s, _itoa_s等。

4、最后对传参错误等问题进行修正，保证功能的正常运行。

经过对示例代码的修复，项目已经能够正常运行HTTP1.0。

5、HTTP报文未完整接收：因为每次recv的结果是一个TCP传输层数据报，可能不是一个完整的HTTP报文，而示例代码仅接收一次就停止接收了，有时可能会不完整，我对其进行了更改，新建一个函数recvData，用来循环接收，判断的标准是如果HTTP头中含有Content-Length字段则接收到该长度，否则接收到一个0接上两组换行\r回车\n就结束，倘若仍没有，则一直接收到服务器关闭。

4.2 对示例代码实现HTTP1.0的解析

示例代码中，首先进入main函数进行初始化的操作，然后循环accept客户的连接请求，并发起新线程ProxyThread处理客户的请求。

在ProxyThread中，首先先recv客户的请求内容，并通过PraseHttpHeader进行处理，得到了HTTP报头，然后对报头内容进行解析，获得了HOST地址和目标端口（80），通过ConnectToServer函数，使用gethostbyname获取到服务器的IP地址，然后connect连接服务器，然后通过send转发用户的请求，最后recv服务器的response，send到用户，最后关闭套接字。

```
BOOL InitSocket();
BOOL ConnectToServer(SOCKET* serverSocket, char* host, Job* j);
void CloseSocket(LPVOID lpParameter, int id);
unsigned int __stdcall ProxyThread(LPVOID lpParameter);
```

4.3 实现HTTP1.1

在示例代码中，仅仅实现了HTTP1.0的功能，但正常的浏览器现今都使用HTTP1.1版本进行通信，因此我又实现了HTTP1.1。

首先创建两个新的函数ProxyThreadServer和ProxyThreadClient负责分别联系和转发客户的目标服务器的报文。

通过对ProxyThread的修改让其在接收到服务器的response之后，发起两个新线程，ProxyThreadServer和ProxyThreadClient，对双方的两个单向通信进行两个分别的“桥接”。由此实现了HTTP1.1。

```
⊞unsigned int __stdcall ProxyThreadClient(LPVOID job){ ... }

⊞unsigned int __stdcall ProxyThreadServer(LPVOID job){ ... }

⊞int ForbidCheck(char* ip, char* host){ ... }
⊞int addIfModifiedHeader(char* buffer, int* size, char* date, Job* j){ ... }
⊞// ...
⊞unsigned int __stdcall ProxyThread(LPVOID job){ ... }
```

4.4 实现套接字的统一管理

为了对套接字进行统一的管理，以及在控制台进行展示，增加了Job数据结构。

在main函数中，每接受一个客户的连接，就会生成一个Job，我给定了最大Job数量=25，在main中，接受客户之前，会检查任务资源池中是否有空余的Job可供分配，分配到资源ID后，就接受客户的连接请求，并在该Job中创建与客户和服务器的套接字及更新相关信息。在ProxyThread中，会传递JID，表示任务ID，通过该ID，线程可以获取到目标Job，并对目标Job进行信息的更新。

任务资源池如下所示：

```
struct  SharedArea
{
    Job jobs[MAX_TCP_CONNECT];
    bool usingjob[MAX_TCP_CONNECT] = { false };
}SHAREDATA;
```

一个Job的数据结构如下所示：

```
struct Job {
    char host[64]; // 表示host的字符串
    char srcIpText[16]; //源IP地址的点分十进制
    ProxyParam socket; //客户和服务器的套接字对
    unsigned long srcIp; //源IP
    unsigned long dstIp; //目的IP
    unsigned long srcPort;//源端口
    unsigned long dstPort;//目的端口
    int jobid; //任务序号
    int status; //任务状态号
    int jobindex; //资源ID（任务ID）
};
```

任务状态号表示的任务状态如下所示：

```
const char* getStatusText(int status) {
    switch (status) {
    case 0: return "等待分配";
    case 1: return "接收用户数据";
    case 2: return "生成HTTP报头";
    case 3: return "建立远程套接字";
    case 4: return "转发用户数据";
    case 5: return "接收服务器数据";
    case 6: return "转发服务器数据";
```

```
    case 7: return "运行HTTP1.1";
    case 8: return "处理完成";
    case 10: return "不支持的请求"; //延时关闭
    default: return "未知";
    }
}
```

4.5 获取第二个控制台

因为在NetProxyServer.exe中，控制台不断输出着各种任务的进行状态，我们需要第二个控制台来查看各种任务连接的状态，因此我使用了《计算机系统》所学的Mapping技术，构造一个新的进程，并与当前进程进行内存共享，进而获得状态信息以及进行控制。

我重新编写了一个控制台程序：NetConsole.exe，用以打印状态信息。

在NetProxyServer.exe的初始化中，增加内存映射和进程调用：

通过对NetProxyServer.exe的内存进行映射共享，构造以下内存共享空间：

```
struct  SharedArea
{
    Job jobs[MAX_TCP_CONNECT];
    bool usingjob[MAX_TCP_CONNECT] = { false };
}SHAREDATA;

struct ControlArea {
    bool stop = false;
}CONTROLDATA;
```

构造的代码如下：

```
hFileHandle = CreateFileMapping(INVALID_HANDLE_VALUE, &sa, PAGE_READWRITE, 0,
sizeof(SharedArea)+ sizeof(ControlArea), (LPCWSTR)L"MyFile");
    printf("hFileHandle=%p\n", hFileHandle);
    if (hFileHandle == NULL) return false;
    //在调用进程的地址空间映射一个文件视图
    sharedMemory = (char*)MapViewOfFile(hFileHandle, FILE_MAP_ALL_ACCESS, 0, 0, 0);
    if (sharedMemory == NULL) return false;
    controlMemory = sharedMemory + sizeof(SharedArea);
    if (controlMemory == NULL) return false;
```

然后通过Mutex实现多进程的数据安全：

```
    //使用互斥对象保证同一时刻只允许一个进程读或写共享内存
    WaitForSingleObject(hMutex, INFINITE);
    memcpy(sharedMemory, &SHAREDATA, sizeof(SharedArea));
    ReleaseMutex(hMutex);
```

构造完成后，自动运行NetConsole.exe，新的进程获取到共享内存的句柄，然后获取到内存信息。

其中，controlMemory是控制信息，sharedMemory是状态信息。

NetConsole.exe和NetProxyServer.exe在controlMemory中共享当前的运行情况，保证同步进行和关闭。

同时NetConsole.exe在后续进行缓存清理的时候，能够强制关闭NetProxyServer.exe。

经过这样的操作，NetConsole.exe就获取到了所有连接的状态信息，并将其打印在第二个控

制台中。

```
printf("%2d %3d %5ld %16s %5ld %16s %-16s %s\n", i, j->jobid, j->dstPort, inet_ntoa(adds),
j->srcPort, j->srcIpText, getStatusText(j->status), j->host);
```

4.6 实现缓存功能

新建文件CacheList.txt和Cache文件夹，其中txt用以保存缓存文件的ID、url、Cookie和Date，Cache文件夹中的每一个文件就是同一个url+cookie的服务器所发的所有反馈的缓存文件。进程每次启动都会读入所有的Cache条目，txt的每一行就是一个Cache条目，包含上述字段并使用“||||”进行分割，初始化时读入到Cache数据结构中。

```
struct CACHE
{
    char url[1024];
    char cookie[10 * 1024];
    char date[50];
    char path[10];
}cache[MAX_CACHE_NUM] = { 0 };
FILE* fcache;
FILE* fcachelist[MAX_CACHE_NUM] = { 0 };
int cachenum = 0;
```

在ProxyThread中，接收到用户请求时会首先调用cacheCheck，检查是否存在url和cookie相同的缓存文件，如果存在则调用addIfModifiedHeader函数，对用户请求的头部添加“If-Modified-Since: ”和cache[i].date 以及**一个换行回车符（\r\n）**，并向服务器转发。接收到服务器的返回信息后，先检查是否返回状态码为304，如果是则直接调用send分段发送缓存文件给客户端。如果不是，则会通过函数cacheCreate创建/修改一个新的/已存在缓存文件列表，并写入到CheckList.txt中，对之后的ProxyThreadServer传递缓存条目编号，每一次接收保存下之后所有的内容作为一个新的缓存文件并追加保存到缓存文件列表中，以期待后续得到304后一次性读取文件列表后直接发送。

4.7 实现客户IP封禁

示例代码在接受用户请求时丢弃了用户的相关信息（未提供指针供存储），而我将此功能完善了，然后就可以获取到用户的IP地址以及端口号。

新建文件ipForbid.txt，每一行都是一个IP地址，程序启动时会进行读入，并保存在封禁IP的数据结构中。

```
int forbidIpNum = 0;
char forbidIp[100][20] = { '\0' };
```

在main函数中，每当接受客户连接请求时，不立即对客户进行封禁，而是按照通常情况，创建一个ProxyThread线程后，对其报文进行分析，将其保存在Job中，以保存到log.txt，记录用户的请求信息后，再调用checkForbid，如果用户被封禁，则拒绝转发，并关闭套接字。

4.8 实现Web资源封禁

新建文件urlForbid.txt，每一行都是一个url或其前缀，程序启动时会进行读入，并保存在封禁url的数据结构中。

```
typedef struct ForbidWeb {
    char url[200];
```

```
    int len;
} FORBIDWEB;
FORBIDWEB forbidWeb[100] = { 0 };
int forbidWebNum = 0;
```

在ProxyThread中，当对用户Http报头分析时，如果发现了用户请求的是封禁的url或以封禁url作为前缀的资源，则拒绝进行转发，并关闭套接字。

4.9 实现钓鱼网站请求拦截与污染

新建文件urlFish.txt，每一行都是一个url或其前缀，程序启动时会进行读入，并保存在钓鱼url的数据结构中。

```
FORBIDWEB tipFishWeb[100] = { 0 };
int tipFishWebNum = 0;
```

在ProxyThread中，当对用户Http报头分析时，如果发现了用户请求的是钓鱼的url或以钓鱼url作为前缀的资源，则拒绝进行转发，并向其转发一个由代理服务器生成的警告网页，提示用户正在访问钓鱼网站，然后关闭套接字。

4.10 实现NetConsole的控制功能

前面我们说过，可以通过内存共享的方式对代理服务器的信息传递到另一个控制台进程并进行打印。下面我们继续实现代理服务器控制台的控制功能，在打印各类连接状态信息的同时，使用kbhit和getch函数，监测用户的键盘输入信息。

```
while (1)
    {
        if (_kbhit())
        {
            doid = (int)_getch();
            switch (doid)
            {
            case B_QUIT:
                proexit(-1);
                break;
            case B_CLEAR_CACHE:
                clearCache();
                break;
            case B_HIDE:
                CONTROLDATA.hidehttps = !CONTROLDATA.hidehttps;
                writeControl();
                    if (CONTROLDATA.hidehttps) {
                        printf("已启用HTTPS静默!\n");
                    }
                    else {
                        printf("已关闭HTTPS静默!\n");
                    }
                    Sleep(1000);
                break;
```

```
            default:
                break;
            }

        }
        Sleep(100);
        refleshCount++;
        if (refleshCount >= REFLESH_TIME) {
            refleshControl();
            refleshShare();
            clearscreen();
            printStatus();
            refleshCount = 0;
        }
    }
```

如果用户输入了Q，则关闭控制台并同时通过共享内存通知代理服务器进行关闭，当代理服务器对数据处理完成后，将会自动退出。同样的方式，也将这项功能加入到代理服务器进程中，如果服务器出错，通过同样的方式，让控制台进程同样退出。

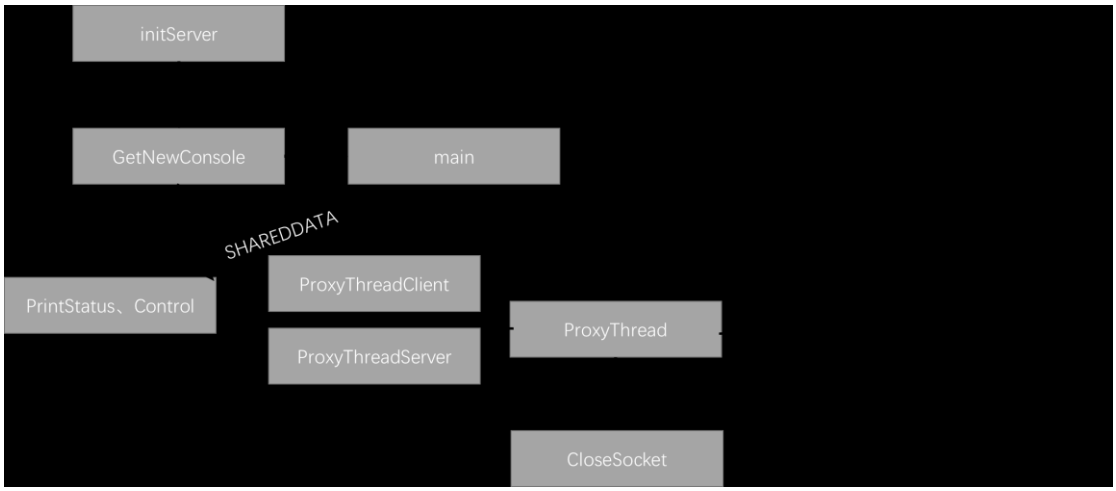
如果用户输入了C，则暂停信息打印，然后询问用户是否确认清除缓存，如果用户再输入Y/y，则通知服务器关闭，待服务器处理数据完成并关闭后，清空CacheList.txt。对于Cache文件夹内的缓存文件无需清空，因为每一次createCache时都使用fopen_s的”w”模式打开，每次都会清空原始内容。

如果用户输入了H，则通知服务器开启/关闭HTTPS静默功能，即对所有HTTPS的报头和分析不做打印输出。

五、测试与联调、编译与发布

在VS2019中，使用CTRL+B快捷键进行编译与生成。
对程序进行调试，测试，修改，完善。
达到要求后，结束并保存。
对每一次功能新增和修改，都使用GitHub Desktop进行commit。

六、项目流程图



实验结果：

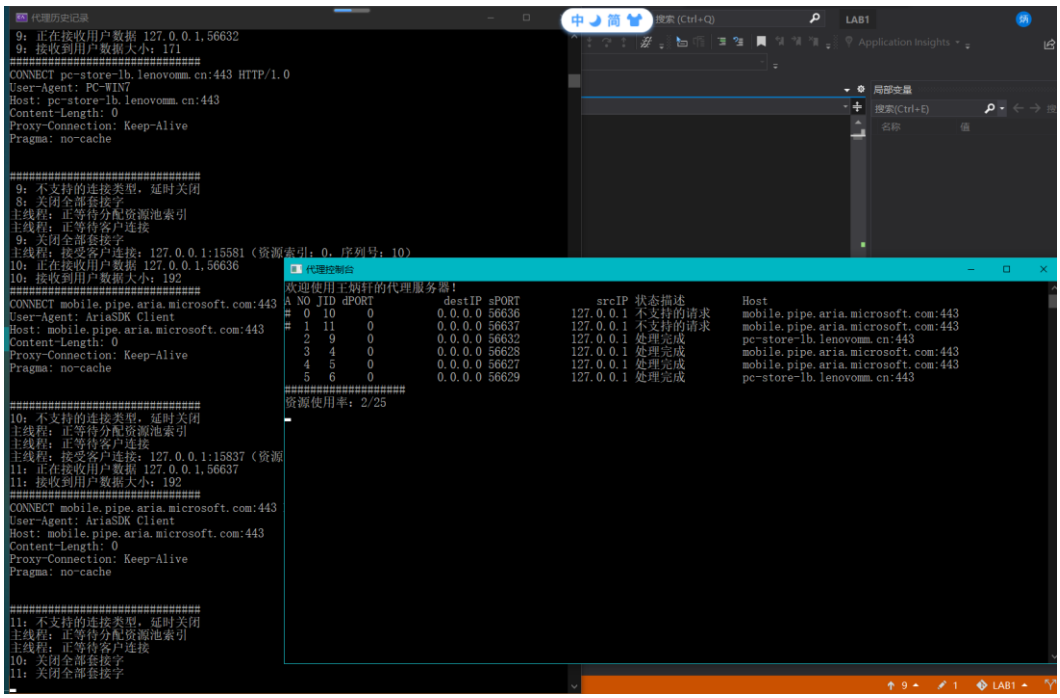
采用演示截图、文字说明等方式，给出本次实验的实验结果。

一、设置计算机代理

打开计算机的代理设置，设置计算机的代理服务器。
为本机(127.0.0.1)的10240端口。

二、启动代理服务器并测试代理

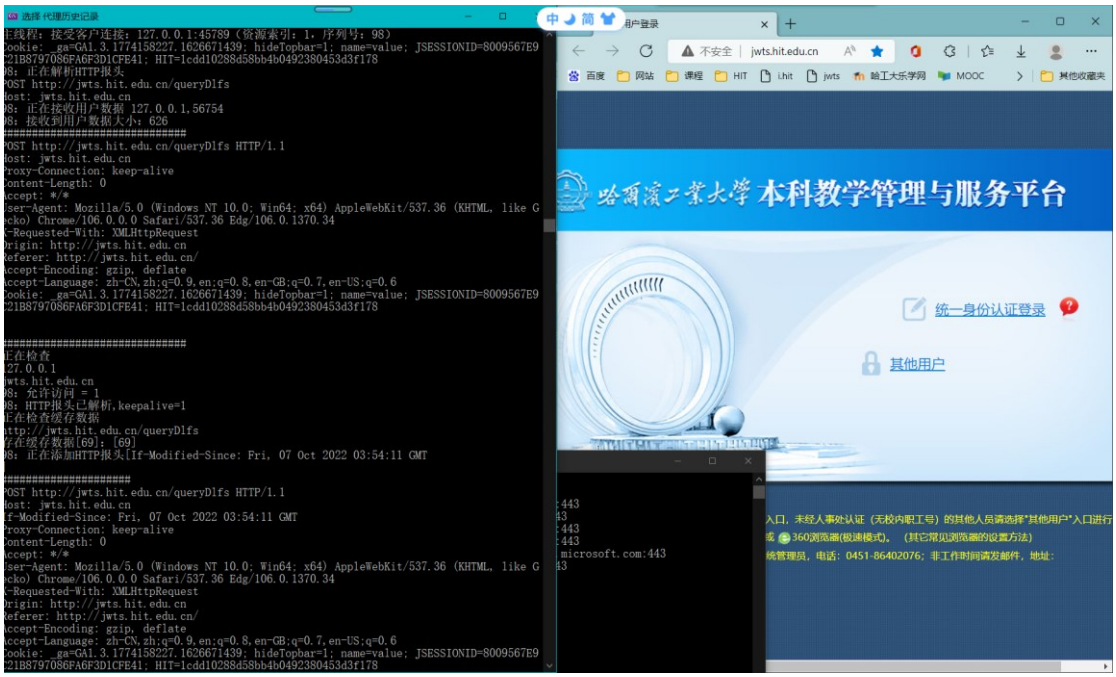
运行代理服务器。结果如下所示。



可以看到，代理服务器正在不断处理来自计算机各进程的请求。

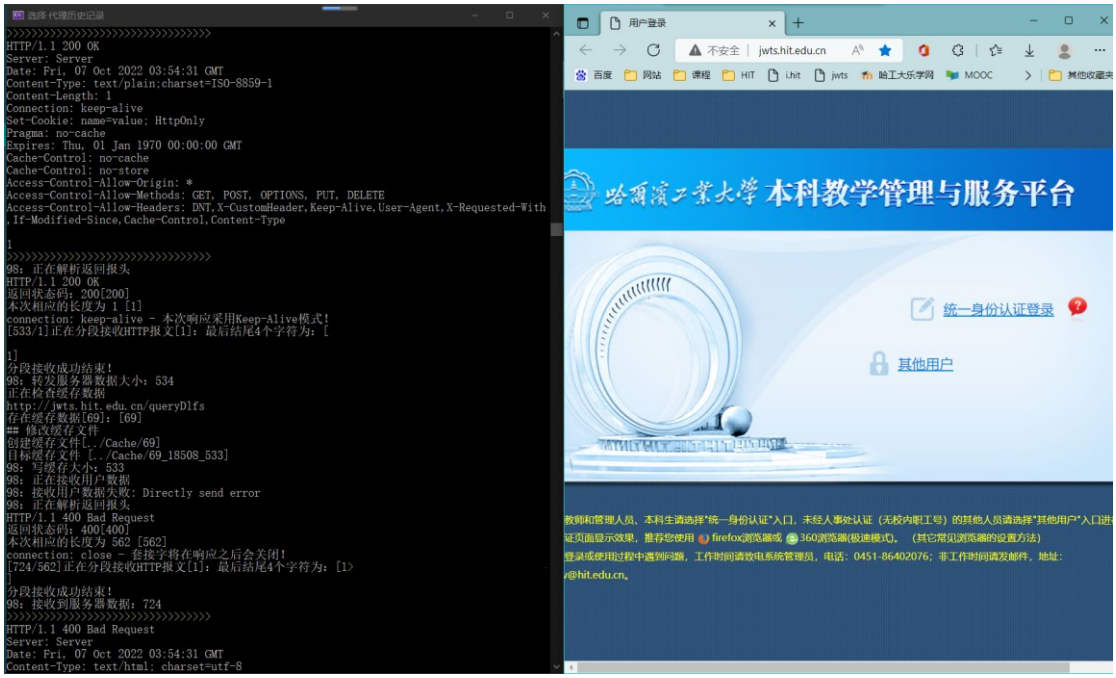
对于HTTPS协议，则拒绝转发，并关闭套接字。

打开浏览器，访问jwts.hit.edu.cn成功进入。同时，代理服务器的历史也打印出了请求信息和响应信息，以及是否命中缓存等。

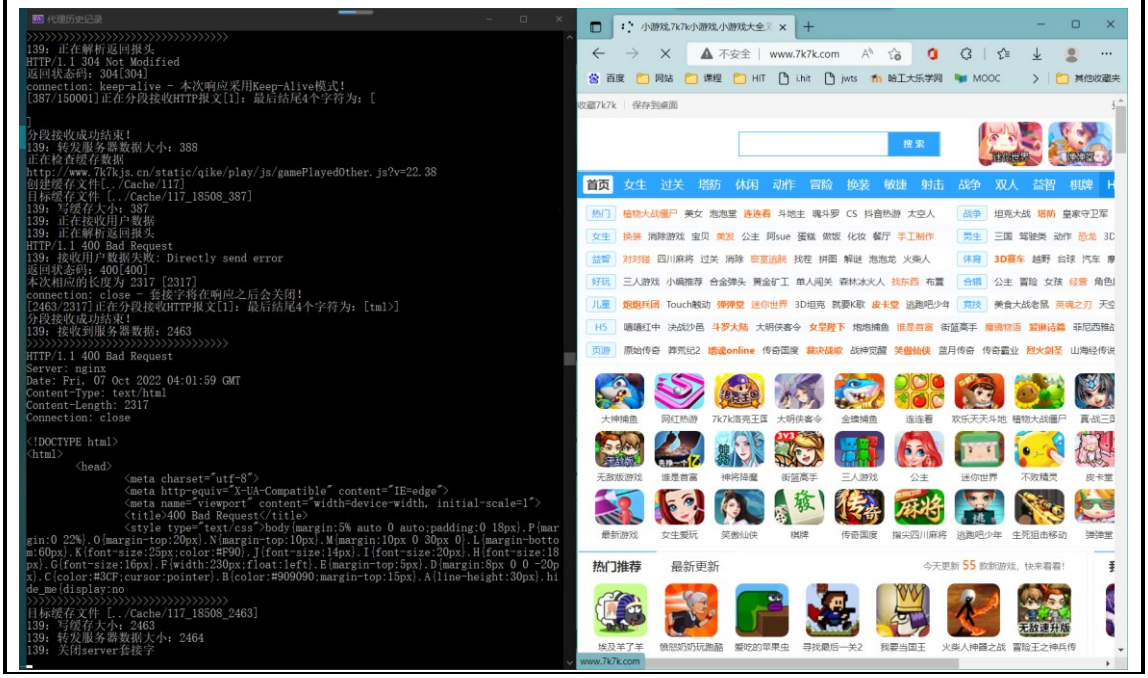


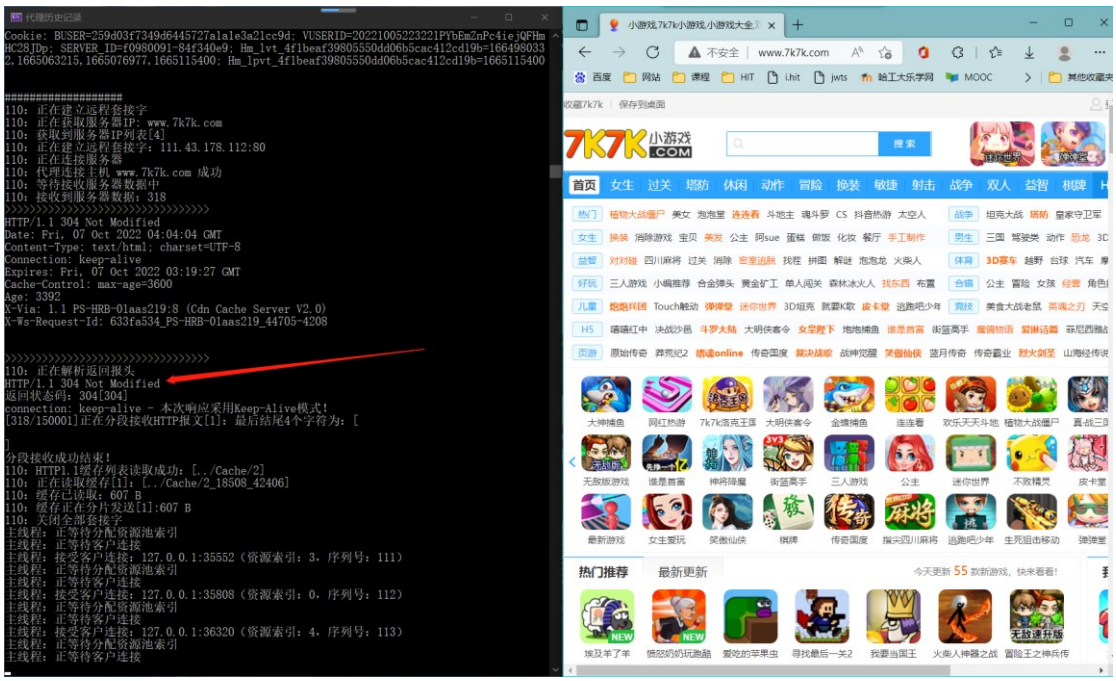
三、测试代理缓存

在浏览器中点击刷新，代理接收到后，对请求头添加If-Modified-Since报头，并转发服务器，得到服务器的响应结果状态码为200。说明服务器对文件进行了更改，由此，重新创建缓存，并发送给客户。浏览器成功打开网页。



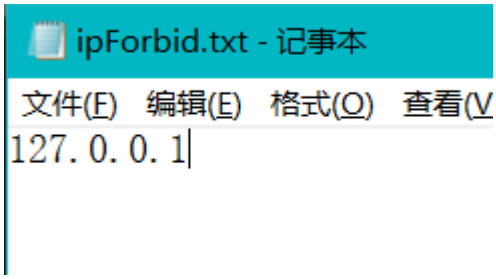
在浏览器中打开另一个网页，再点击刷新，代理接收到后，对请求头添加If-Modified-Since报头，并转发服务器，得到服务器的响应结果状态码为304。说明服务器未对文件进行更改，由此，读取缓存，并发送给客户。浏览器成功打开网页。



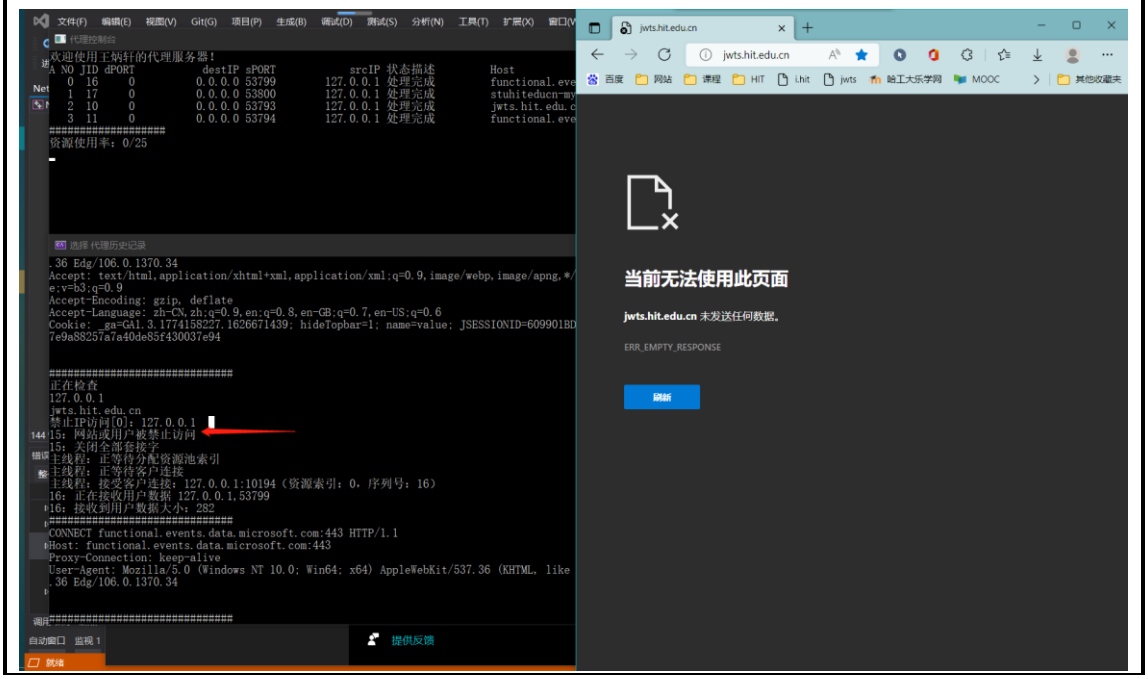


四、测试IP封禁

在ipForbid.txt中添加本机IP，重启代理服务器进行测试。

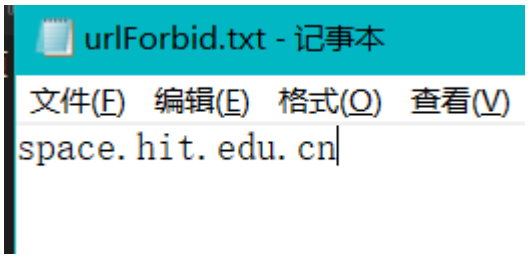


控制台打印出发现了封禁IP，所有由本机发送的请求都不会被转发。

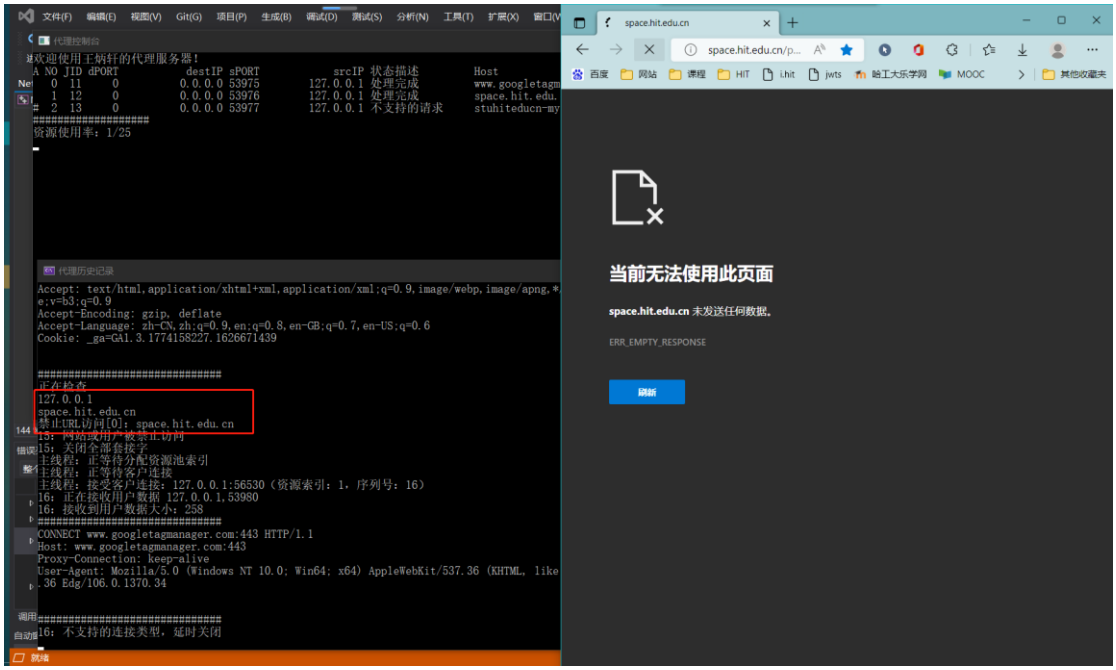


五、测试网站封禁

在urlForbid.txt中添加一个禁止访问的网站，重启代理服务器进行测试。

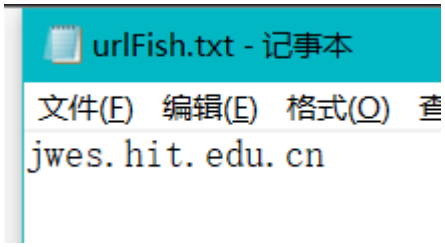


控制台发现访问封禁URL及其下的资源，请求都不会被转发。

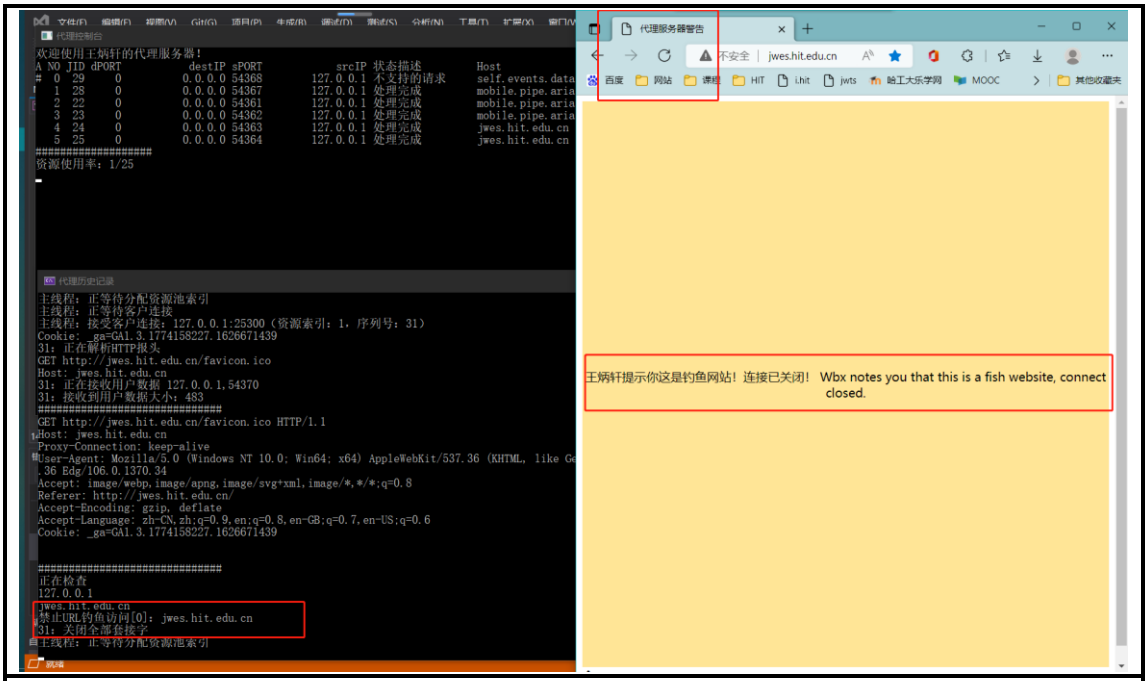


六、测试网站钓鱼污染

在urlFish.txt中添加一个钓鱼网站，重启代理服务器进行测试。



控制台发现访问封禁URL及其下的资源，请求都会被拦截并相应返回一个模拟网站。



问题讨论：

1、本次实验所给的示例代码不仅各种错误，且仅能够完成HTTP1.0的功能，通过对其进行修复升级，实现了HTTP1.1的功能。

2、本次实验仅要求了部分的HTTP协议，如以下情况，则用户无法访问：

对于HTTPS协议（目标端口443），拒绝进行转发。

对于除POST/GET的其他HTTP请求方法，如CONNECT，拒绝进行转发。

心得体会：

结合实验过程和结果给出实验的体会和收获。

本次实验由于教务处的突然提前一周上课，导致实验时间非常仓促，我经过阅读实验指导书，到完成代码，这几天一直都在做这件事。况且，实验的示例代码有如此多的错误，以及各种的内存溢出和非法访问等问题，**极大地影响了计算机的性能和稳定性，导致运行时可能出现系统蓝屏、DLL丢失，导致系统强制重启，丢失所有未保存的工作！我希望能对指导书的实验代码进行更新。**

本次实验也让我深入地学会了SOCKET编程、使用winsockAPI函数和对HTTP报文结构的深入解析，让我学会了在今后的项目中使用更加底层的计算机网络模块。