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	实验 1：HTTP 代理服务器的设计与实现				
姓名		院系			
班级		学号			
任课教师	刘亚维	指导教师	刘亚维		
实验地点	G002	实验时间	2024.4.16		
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					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

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。

(3) 扩展 HTTP 代理服务器，支持如下功能：

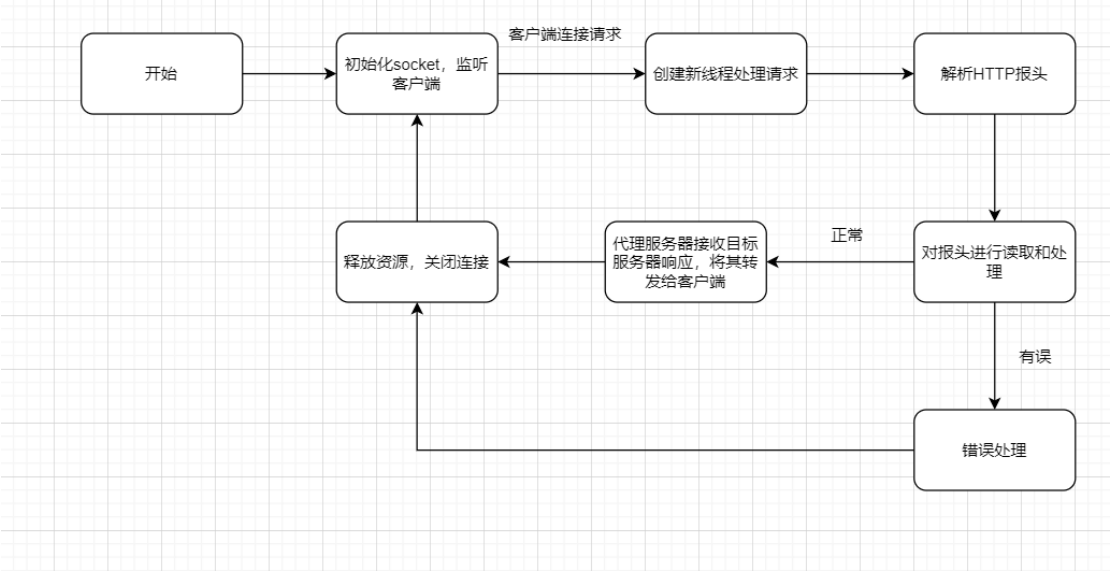
a) 网站过滤：允许/不允许访问某些网站；

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：

整个程序的基本流程如下图所示



1)主函数：main函数首先对套接字库初始化，绑定端口，无限循环监听客户连接请求。当监听到时，调用accept函数接收这个请求，读取地址，创建新线程处理请求。处理完毕之后睡眠以供其他线程执行。退出循环后清理套接字

```
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;//调用 InitSocket 函数来初始化套接字库，创建代理服务器的套接字，并绑定端口。
    ProxyParam* lpProxyParam;
    HANDLE hThread;
    DWORD dwThreadID;
    //代理服务器不断监听
    while(true){
start:
        acceptSocket = accept(ProxyServer,NULL,NULL);//调用 accept 函数接收客户端的连接请求。接收到的连接请求对应的客户端套接字为 acceptSocket
        lpProxyParam = new ProxyParam;//c++下使用new申请空间
        if(lpProxyParam == NULL){
            continue;
        }
        SOCKADDR_IN acceptAddr;
        int addrlen = sizeof(SOCKADDR);
        acceptSocket = accept(ProxyServer, (SOCKADDR*)&acceptAddr, &addrlen);//再次调用 accept 函数接收客户端的连接请求，这次会获取客户端的地址信
        printf("\n用户IP地址为%s\n", inet_ntoa(acceptAddr.sin_addr));
        //用户过滤
```

```
        //用户过滤
        switch (is_permit)
        {
        case 0:
            if(strcmp(inet_ntoa(acceptAddr.sin_addr), my_ip) == 0){
              printf("您的主机已被屏蔽！\n");
              goto start;
            }
            break;
        case 1:
            break;
        }
        lpProxyParam->clientSocket = acceptSocket;//将接收到的客户端套接字 acceptSocket 存储到 lpProxyParam 结构体中。
        hThread = (HANDLE)_beginthreadex(NULL, 0, &ProxyThread,(LPVOID)lpProxyParam, 0, 0);//创建一个新的线程来处理这个客户端的连接请求，线程的
        CloseHandle(hThread);
        Sleep(200);//让当前线程暂停 200 毫秒，为其他线程提供执行的机会。
    }
    closesocket(ProxyServer);
    WSACleanup();//当代理服务器需要关闭时，关闭代理服务器的套接字，并清理套接字库。
    return 0;
}
//*********************************
```

2）初始化套接字库：加载套接字库，使用sin_family，sin_port,sin_addr等设置代理服务器的地址族、端口号和连接请求

```
145     ProxyServer= socket(AF_INET, SOCK_STREAM, 0);//调用 socket 函数创建一个 TCP 套接字，用于监听客户端的连接请求。
146     if(INVALID_SOCKET == ProxyServer){
147         printf("创建套接字失败，错误代码为：%d\n",WSAGetLastError());
148         return FALSE;
149     }
150     ProxyServerAddr.sin_family = AF_INET;//设置代理服务器的地址族为 AF_INET，表示使用 IPv4 地址。
151     ProxyServerAddr.sin_port = htons(ProxyPort);//指向网络字节顺序的端口号
152     ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;//设置代理服务器的 IP 地址为 INADDR_ANY，表示接收所有的客户端连接请求。
153     if(bind(ProxyServer,(SOCKADDR*)&ProxyServerAddr,sizeof(SOCKADDR)) == SOCKET_ERROR){
154         printf("绑定套接字失败\n");
155         return FALSE;
156     }
157     if(listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR){
158         printf("监听端口%d 失败",ProxyPort);
159         return FALSE;
160     }
161         return TRUE;
162 }
```

3）线程主体：首先使用recv从目标服务器处截获报文存储在缓冲区buffer中以供解析，然后调用解析报文函数parseHTTPHead解析报头并存储在新声明的结构体HTTPHeader中，接下来根据报头的信息完成附加功能，后面详细叙述。接下来将客户端的请求报文发送给目标服务器，并接收传送回来的报文，判断缓存信息后，转发给客户端。这其中如果出现错误则使用goto函数跳转到错误处理位置，即关闭客户端和服务器端套接字，释放线程，休眠并返回。

```
unsigned int __stdcall ProxyThread(LPVOID lpParameter){
    char Buffer[MAXSIZE];//存储报文
    char fill_buffer[MAXSIZE];//存储缓存数据
    char* CacheBuffer;//存储cache缓存
    char* DataBuffer;//为构建date字段创建缓存
    ZeroMemory(Buffer,MAXSIZE);//初始化
    SOCKADDR_IN clientAddr;//存储客户端地址
    int length = sizeof(SOCKADDR_IN);
    int recvSize;
    int ret;
    FILE* file;//
    char field[] = "Date";//下一步捕捉date字段使用
    char data_save[30];//保存字段date的值
    ZeroMemory(data_save, 30);
    ZeroMemory(fill_buffer, MAXSIZE);//初始化
    char filename[100];//存储缓存文件名字
    ZeroMemory(filename, 100);

    recvSize = recv(((ProxyParam *)lpParameter)->clientSocket,Buffer,MAXSIZE,0);//从网页截获报文存储至buffer中
    HttpHeader* httpHeader = new HttpHeader;//创建一个 HttpHeader 结构体，用于存储 HTTP 头部信息。
    if(recvSize <= 0){
        goto error;
    }//goto语句后不能再声明变量
    //printf("\n======截获的报文数据是======\n%s\n",Buffer);

```

```
CacheBuffer = new char[recvSize + 1];
ZeroMemory(CacheBuffer,recvSize + 1);
memcpy(CacheBuffer,Buffer,recvSize);//将从报文读到的内容写入我自己的缓存中
ParseHttpHead(CacheBuffer,httpHeader);//解析报文
// printf("\n======httpHeader是======\n%s\n",httpHeader);
// printf("\n======CacheBuffer是======\n%s\n",CacheBuffer);
DataBuffer = new char[recvSize + 1];
ZeroMemory(DataBuffer,recvSize + 1);

file_name(httpHeader->url, filename);//构造文件名

//更新缓存
if((file = fopen(filename, "rb") )!= NULL){
    printf("\n代理服务器在该url下有相应缓存！\n");
    fread(fill_buffer, sizeof(char), MAXSIZE, file);
    fclose(file);
    get_date(fill_buffer, field, data_save);
    printf("date_str: %s\n", data_save);
    modify_cache(Buffer, data_save);
    printf("\n======改造后的请求报文======\n%s\n", Buffer);
    have_cache = true;
}
```

```
//禁止访问的网站
if(strstr(httpHeader->url, block_web) != NULL)
{
    printf("禁止访问\n");
    goto error;
}

//钓鱼网站
if(strstr(httpHeader->url, fish_web) != NULL)
{
    printf("钓鱼网站\n");
    printf("钓鱼成功：您所前往的%s已被引导至%s\n",fish_web,target_web);
    memcpy(httpHeader->host, target_web, strlen(target_web)+1);
    printf("host: %s\n", httpHeader->host);
    memcpy(httpHeader->url, target_url, strlen(target_url)+1);
    printf("url: %s\n", httpHeader->url);
}
delete CacheBuffer;//释放缓存

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);
printf("\n======发送的请求报文是======\n%s\n",Buffer);
```

```
//等待目标服务器返回数据
recvSize = recv(((ProxyParam *)lpParameter)->serverSocket,Buffer,MAXSIZE,0);
printf("\n======返回的请求报文是======\n%s\n",Buffer);

if(recvSize <= 0){
    goto error;
}
printf("have_cache: %d\n", have_cache);
printf("need_cache: %d\n", need_cache);
//有缓存时，判断返回的状态码是否为304，是则将缓存数据发送给客户端
printf("\n======代理服务器接收到数据======\n%s\n",Buffer);
printf("网站url是：%s\n", httpHeader->url);
if(have_cache == true){
    read_cache(Buffer, httpHeader->url);//读取缓存数据
}
if(need_cache == true){
    buide_cache(Buffer, httpHeader->url);//将数据写入缓存
}
//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);
printf("\n======客户端接收到数据======\n%s\n",Buffer);
```

```
266         //错误处理
267         error:
268             printf("关闭套接字\n");
269             Sleep(200);
270             closesocket(((ProxyParam*)lpParameter)->clientSocket);
271             closesocket(((ProxyParam*)lpParameter)->serverSocket);
272             free(lpParameter);
273             _endthreadex(0);
274             return 0;
275     }
```

4）解析HTTP报文:使用strtok_s提取报文的第一行存储到指针p中，判断p指向的第一个字符从而判断是GET方式还是POST方式，接下来提取第二行填充主机名、cookie等，最后继续提取。完成后，httpheader中有需要的报头的所有信息

```
void ParseHttpHead(char *buffer,HttpHeader * httpHeader){
    char *p;
    char *ptr;
    const char * delim = "\r\n";//分隔符，处理是跳过
    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);//填充httpHeader结构体
    }
    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);//提取第二行
```

```
    while(p){
        switch(p[0]){
            case 'H'://Host
                memcpy(httpHeader->host,&p[6],strlen(p) - 6);//主机名
                break;
            case 'C'://Cookie
                if(strlen(p) > 8){
                    char header[8];
                    ZeroMemory(header,sizeof(header));
                    memcpy(header,p,6);//提取cookie
                    if(!strcmp(header,"Cookie")){
                        memcpy(httpHeader->cookie,&p[8],strlen(p) -8);
                    }
                }
                break;
            default:
                break;
        }
        p = strtok_s(NULL,delim,&ptr);//继续提取
    }
}
```

5）连接目标服务器：设置地址族、端口号，获取主机信息和ip地址，将点分十进制的ip地址转换为网络字节序格式，然后创建套接字，调用connect连接到目标服务器

```
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;//设置地址族为 AF_INET，表示使用 IPv4 地址。
    serverAddr.sin_port = htons(HTTP_PORT);//设置端口号为 HTTP 服务的端口，htons 函数用于将主机字节序转换为网络字节序。
    HOSTENT *hostent = gethostbyname(host);//调用 gethostbyname 函数获取主机的信息，返回一个 HOSTENT 结构体指针。
    if(!hostent){
        return FALSE;
    }
    in_addr Inaddr=*( (in_addr*) *hostent->h_addr_list);//获取主机的 IP 地址。
    serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));//将 IP 地址的格式从点分十进制格式转换为网络字节序格式，并设置为目标服务器的 IP 地址
    *serverSocket = socket(AF_INET,SOCK_STREAM,0);//调用 socket 函数创建一个套接字。
    if(*serverSocket == INVALID_SOCKET){
        return FALSE;
    }
    if(connect(*serverSocket,(SOCKADDR *)&serverAddr,sizeof(serverAddr)) == SOCKET_ERROR){//调用 connect 函数连接到目标服务器
        closesocket(*serverSocket);
        return FALSE;
    }
    return TRUE;
```

8）缓存和添加if-modified-since头行：这部分在线程函数中实现，首先根据目标服务器的url，调用函数file_name创建文件名。为构建文件名单独封装一个函数的意义是，确保访问相同的网络资源时可以访问相同的缓存文件，而且保证文件名不出现非法字符。

```
    CacheBuffer = new char[recvSize + 1];
    ZeroMemory(CacheBuffer,recvSize + 1);
    memcpy(CacheBuffer,Buffer,recvSize);//将从报文读到的内容写入我自己的缓存中
    ParseHttpHead(CacheBuffer,httpHeader);//解析报文
    // printf("\n======httpHeader是======\n%s\n",httpHeader);
    // printf("\n======CacheBuffer是======\n%s\n",CacheBuffer);
    DataBuffer = new char[recvSize + 1];
    ZeroMemory(DataBuffer,recvSize + 1);

    file_name(httpHeader->url, filename);//构造文件名

    //更新缓存
    if((file = fopen(filename, "rb") )!= NULL){
        printf("\n代理服务器在该url下有相应缓存！\n");
        fread(fill_buffer, sizeof(char), MAXSIZE, file);
        fclose(file);
        get_date(fill_buffer, field, data_save);
        printf("date_str: %s\n", data_save);
        modify_cache(Buffer, data_save);
        printf("\n======改造后的请求报文======\n%s\n", Buffer);
        have_cache = true;
    }
```

```
/*
函数名：file_name
输入：网站url，文件指针
输出：空
功能：根据url编辑保存文件的文件名
*/
void file_name(char* url, char* filename){
    int count = 0;//计算已经提取的字符数量。
    while (*url != '\0') {
        if ((*url >= 'a' && *url <= 'z') || (*url >= 'A' && *url <= 'Z') || (*url >= '0' && *url <= '9')) {
            *filename++ = *url;
            count++;//如果当前字符是字母或数字，将其添加到文件名中，并增加计数器
        }
        if (count >= 95)//文件名长度限制为95个字符
            break;
        url++;
    }
    *filename = '\0';  // 添加字符串结束符
}
```

接下来尝试打开根据url得到的文件名，如果存在则说明该网页已缓存，从文件中读取缓存报文，调用get_date函数获得日期字段，然后调用modify_cache函数添加if-modified-since头行。

```
/*
    函数名：get_date
    输入：缓冲区，字段名，保存字段值的指针
    输出：布尔值，成功获得则返回true
    功能：分析http报文头部的field字段，获取时间
*/
bool get_date(char* buffer, char* field, char* date){
    char* p;
    char* ptr;
    char* temp_buffer;
    char* temp;
    temp_buffer = (char*)malloc(MAXSIZE*sizeof(char));//为临时缓冲区分配内存
    ZeroMemory(temp_buffer, MAXSIZE);
    ZeroMemory(date, 30);//初始化
    memcpy(temp_buffer, buffer, strlen(buffer));//将buffer中的内容复制到temp_buffer中
    p = strtok_s(temp_buffer, "\r\n", &ptr);//分割报文,获取第一行
    while(p != NULL){
        if(strstr(p, field) != NULL){//使用strstr函数查找field字段，即查找date字段
            memcpy(date, &p[strlen(field) + 2], strlen(p) - strlen(field) + 2);//获取时间
            return true;
        }
        p = strtok_s(NULL, "\r\n", &ptr);//继续分割
    }
    return false;
}
```

```
/*
    函数名：modify_cache
    输入：字段名，保存字段值的指针
    输出：空
    功能：用于第二次访问网站且无需修改时，这时直接从缓存中读取文件，需要向其插入If-Modified-Since字段
*/
void modify_cache(char* buffer, char* date){
    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 (*date != '\0') {//插入对象文件的最新被修改时间
        *pos++ = *date++;
    }
    *pos++ = '\r';
    *pos++ = '\n';
    for (i = 0; i < strlen(temp); i++) {
        *pos++ = temp[i];
    }
```

最后，通过比较need_cache和have_cache两个标志位决定新建缓存还是从缓存中读取。前者初始真，在read_cache中修改；后者初始为假，在build_cache中和更新缓存时修改。

```
    printf("have_cache: %d\n", have_cache);
    printf("need_cache: %d\n", need_cache);
    //有缓存时，判断返回的状态码是否为304，是则将缓存数据发送给客户端
    printf("\n======代理服务器接收到数据======\n%s\n",Buffer);
    printf("网站url是：%s\n", httpHeader->url);
    if(have_cache == true){
        read_cache(Buffer, httpHeader->url);//读取缓存数据
    }
    if(need_cache == true){
        buide_cache(Buffer, httpHeader->url);//将数据写入缓存
    }
```

```
    输入：缓冲区，网站url
    输出：空
    功能：构造cache,即读取报文并写入文件
*/
void buide_cache(char* buffer, char* url){
    char* p;
    char* ptr;
    char* temp_buffer;
    char num[10];
    temp_buffer = (char*)malloc(MAXSIZE*sizeof(char));
    ZeroMemory(temp_buffer, MAXSIZE);
    ZeroMemory(num, 10);
    memcpy(temp_buffer, buffer, strlen(buffer));
    p = strtok_s(temp_buffer, "\r\n", &ptr);
    if(strstr(p, "200") != NULL){//状态码为200时需要进行缓存/更新
        have_cache = true;
        char filename[100];
        file_name(url, filename);
        FILE* file;
        errno_t err = fopen_s(&file, filename, "wb");//打开文件,记录错误信息
        if (err || file == NULL) {
            printf("\nCannot open file %s. Error code: %d\n", filename, err);
            return;
        }
        fwrite(buffer, sizeof(char), strlen(buffer), file);
        fclose(file);
    }
    printf("\n===============缓存构建成功===============\n");
}
```

```
/*
    函数名：read_cache
    输入：缓冲区，文件名
    输出:空
    功能：从cache中获得报文，即从文件中读数据
*/
void read_cache(char* buffer, char* filename){
    char* p;
    char* ptr;
    char* temp_buffer;
    char num[10];
    temp_buffer = (char*)malloc(MAXSIZE*sizeof(char));
    ZeroMemory(temp_buffer, MAXSIZE);
    ZeroMemory(num, 10);
    memcpy(temp_buffer, buffer, strlen(buffer));
    p = strtok_s(temp_buffer, "\r\n", &ptr);
    if(strstr(p, "304")!= NULL){//状态码为304时可以直接读取缓存数据而不必建立网络连接
        FILE* file ;
        fopen_s(&file, filename, "rb");
        // if (file == NULL) {
        //     perror("文件打开失败\n");
        //     return;
        // }
        fread(buffer, sizeof(char), MAXSIZE, file);
        fclose(file);
        need_cache = false;//从缓存中读取数据，不需要再次缓存
        printf("\n===============缓存读取成功===============\n");
    }
}
```

9）扩展功能

a) 网站过滤：允许/不允许访问某些网站：在线程主体函数中实现。在获得目标服务器URL后，将其与已经过滤的服务器URL比对，如果存在字串关系则跳转到错误函数。未被过滤掉网站可以正常访问

```
//禁止访问的网站
if(strstr(httpHeader->url, block_web) != NULL)
{
    printf("禁止访问\n");
    goto error;
}
```

b) 用户过滤：支持/不支持某些用户访问外部网站：在main函数中实现。is_permit控制是否开启用户过滤功能；开启后，使用acceptAddr.sin_addr获取主机ip，如果与设定的过滤ip一致则屏蔽，跳转到while函数开头的部分

```
//用户过滤
switch (is_permit)
{
case 0:
    if(strcmp(inet_ntoa(acceptAddr.sin_addr), my_ip) == 0){
      printf("您的主机已被屏蔽！\n");
      goto start;
    }
    break;
case 1:
    break;
}
```

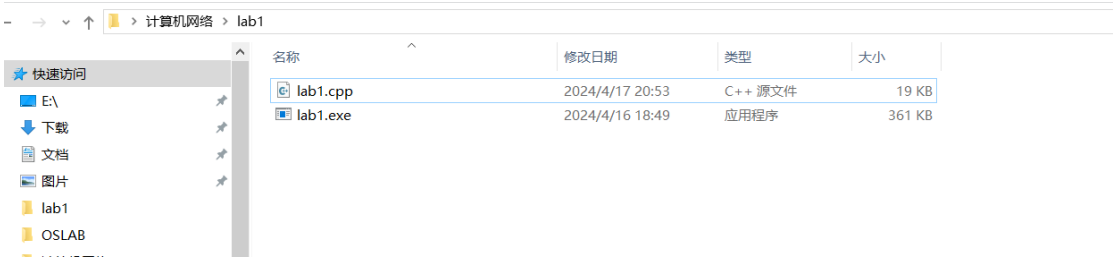
c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）：在线程主体函数中实现。在获得目标服务器URL后，将其与钓鱼网站URL比对，如果存在字串关系则修改存储的httpheader的内容，将主机名和URL都修改为钓鱼目的网站的信息，完成钓鱼任务。

```
225     //钓鱼网站
226     if(strstr(httpHeader->url, fish_web) != NULL)
227     {
228         printf("钓鱼网站\n");
229         printf("钓鱼成功：您所前往的%s已被引导至%s\n",fish_web,target_web);
230         memcpy(httpHeader->host, target_web, strlen(target_web)+1);
231         printf("host: %s\n", httpHeader->host);
232         memcpy(httpHeader->url, target_url, strlen(target_url)+1);
233         printf("url: %s\n", httpHeader->url);
234     }
```

实验结果：

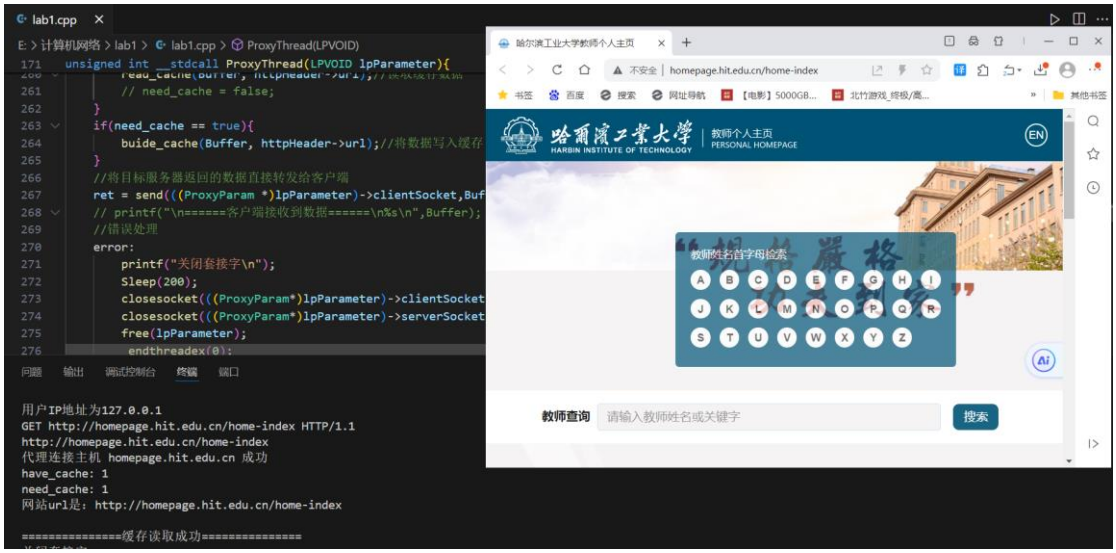
1.修改代理，将代理服务器地址设置为127.0.0.1，端口设置为7890，打开www.baidu.com，发现不能连接

2. 切换到当前程序文件夹下运行程序，在程序运行前文件夹没有缓存文件

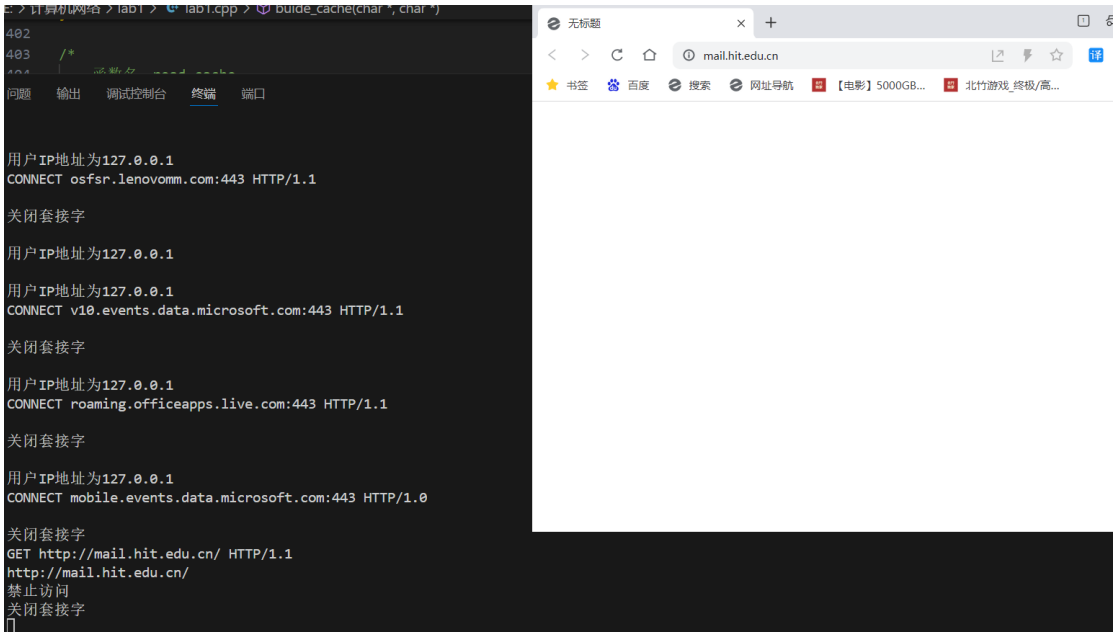


3. 访问hompage.hit.edu.cn，网站成功打开，文件夹下出现缓存文件

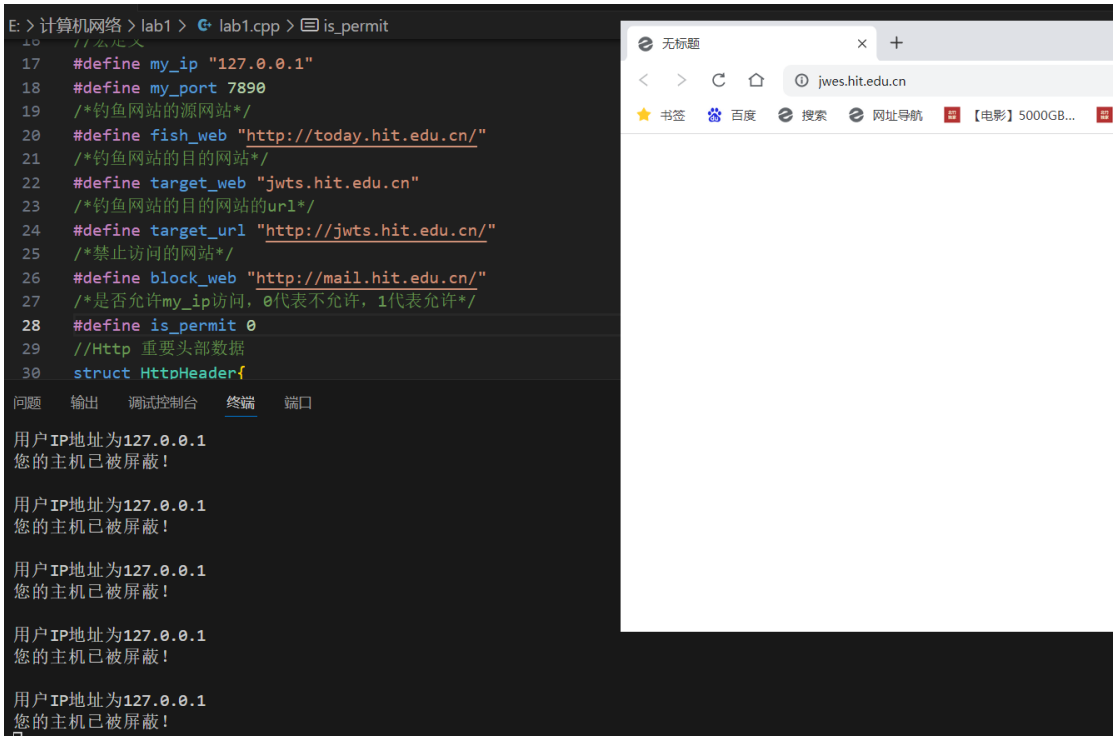
3.再次访问homepage.hit.edu.cn，检测到本地缓存，为请求报文添加if-modified-since头行，接收返回报文，如果返回报文中有304，则从本地读取缓存



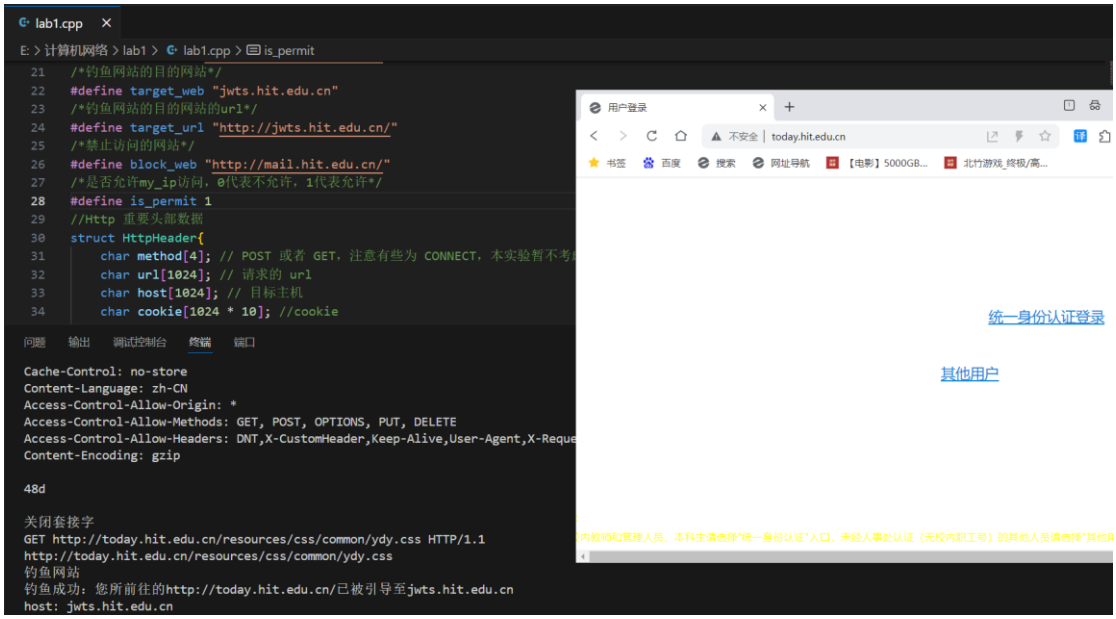
4.访问过滤网站mail.hit.edu.cn,不能返回数据，控制台打印禁止访问



5.将is_permit 设置为0重新运行，过滤用户127.0.0.1不能访问网站，控制台打印IP和屏蔽信息



6. 访问钓鱼网站：访问today.hit.edu.cn，控制台打印钓鱼成功信息，网页跳转到jwts.hit.edu.cn的登陆界面



问题讨论：

（1）Socket 编程的客户端和服务器端主要步骤；

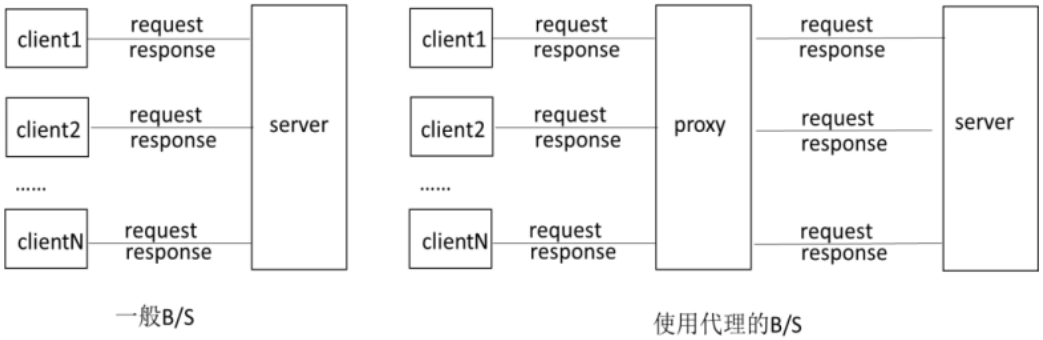
a) 客户端

- 初始化套接字库
- 创建Socket
- 向服务器发出连接请求
- 连接建立后，向服务器请求数据，并置于等待状态，等待服务器返回数据
- 关闭连接
- 关闭套接字库

b) 服务端

- 初始化套接字库
- 创建套接字
- 绑定套接字
- 监听端口
- 接受连接请求，返回新的套接字
- 接受客户端请求消息，返回请求数据，进行通信
- 关闭套接字
- 关闭套接字库

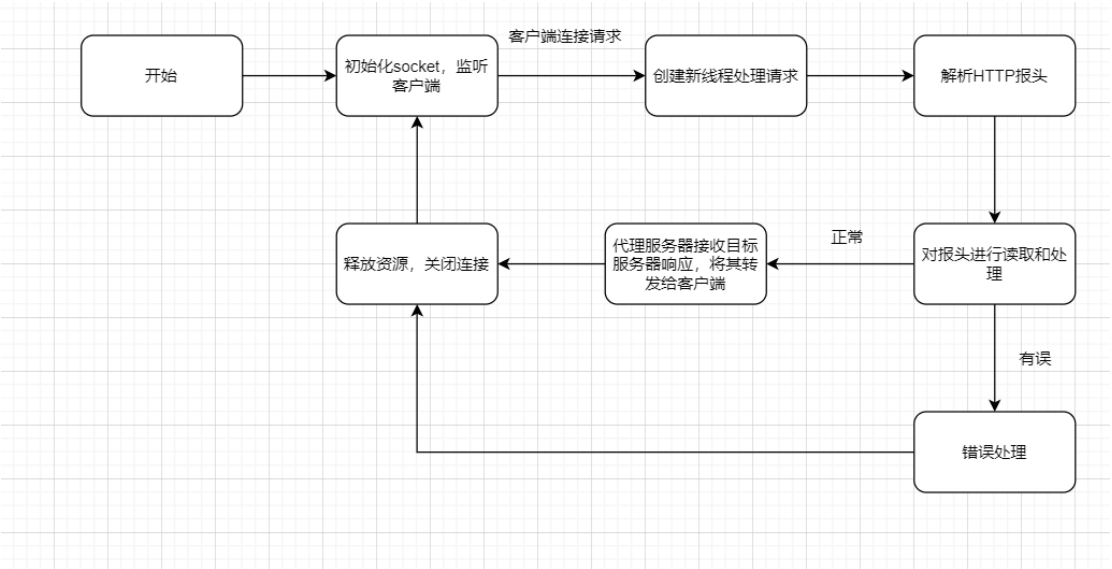
（2）HTTP 代理服务器的基本原理；



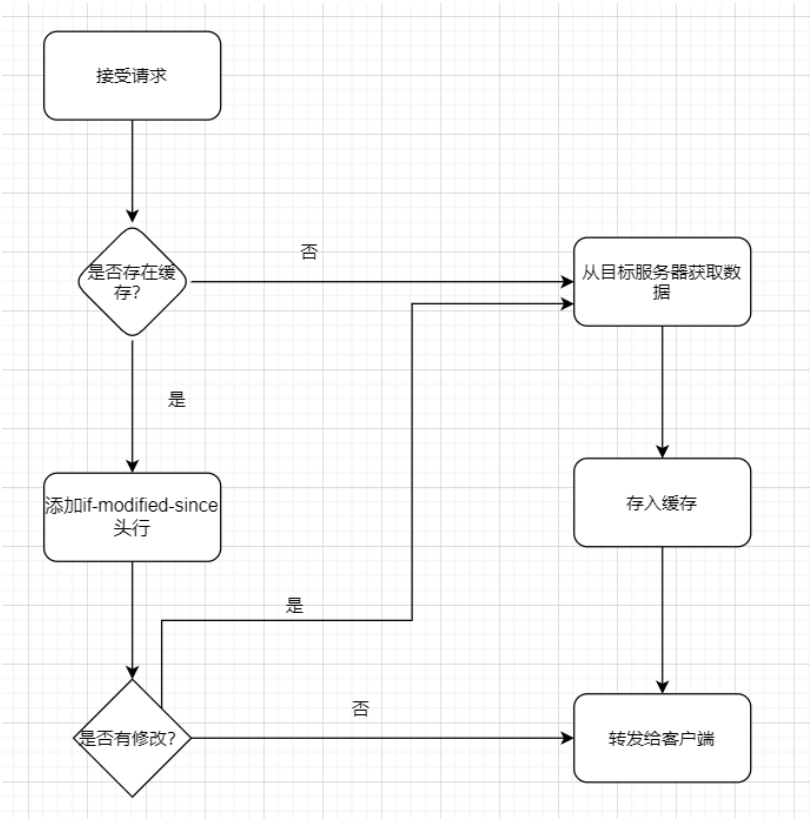
代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。代理服务器在指定端口（例如 8080）监听浏览器的访问请求（需要在客户端浏览器进行相应的设置），接收到浏览器对远程网站的浏览请求时，代理服务器开始在代理服务器的缓存中检索 URL 对应的对象（网页、图像等对象），找到对象文件后，提取该对象文件的最新被修改时间；代理服务器程序在客户的请求报文首部插入<If-Modified-Since: 对象文件的最新被修改时间>，并向原 Web 服务器转发修改后的请求报文。如果代理服务器没有该对象的缓存，则会直接向原服务器转发请求报文，并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。

（3）HTTP 代理服务器的程序流程图；

本实验程序可以概括为下图



在添加上扩展功能和缓存功能后，代理服务器的功能可以概括为下图



（4）实现 HTTP 代理服务器的关键技术及解决方法

可以分为单用户代理服务器和多用户代理服务器两步进行

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。

（5）HTTP 代理服务器实验验证过程以及实验结果；

见上文

（6）HTTP 代理服务器源代码（带有详细注释）

受限于篇幅，见附件

心得体会：

本次实验实现了一个简易的HTTP 代理服务器，通过调试实验指导书给定的代码就可以完成基本功能；在深入了解套接字编程、多线程编程技术和部分c++语法规则后，通过封装函数的形式完成了扩展功能。总的来说实验难度剧中，我本人大量时间耗费在调试返回报文、文件读写的等内容上，一开始接收不到返回304 NOT MODIFY的报文，后来又出现文件打开失败的错误，最

终调试成功。