華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

LAB REPORT

计算机通信与网络实验报告

姓名 邹雅

学号 U201915035

班级 ACM1901

时间 2021 年 12 月 18 日

目录

1 实验一 Socket 编程实验

1.1 环境

1.1.1 开发运行平台

开发及运行设备配置:

CPU：2.3 GHz Quad-Core Intel Core i5

设备 OS：macOS BigSur 11.3.1

Memory：8 GB 2133 MHz LPDDR3

虚拟机：Windows 10 64-bit

开发平台：Visual Studio 2019

编程语言：C++

1.2 系统功能需求

编写一个 Web 服务器软件，要求如下:

基本要求

- 可配置 Web 服务器的监听地址、监听端口和主目录 (不得写在代码里面，不能每配置一次都要重编译代码);
- 能够单线程处理一个请求。当一个客户 (浏览器, 输入 URL:http://202.103.2.3/index.html) 连接时创建一个连接套接字;
- 从连接套接字接收 http 请求报文，并根据请求报文的确定用户请求的网页文件;
- 从服务器的文件系统获得请求的文件。创建一个由请求的文件组成的 http 响应报文;
- 经 TCP 连接向请求的浏览器发送响应，浏览器可以正确显示网页的内容;

高级要求

- 能够传输包含多媒体 (如图片) 的网页给客户端，并能在客户端正确显示;
- 在服务器端的屏幕上输出请求的来源 (IP 地址、端口号和 HTTP 请求命令行);
- 在服务器端的屏幕上能够输出对每一个请求处理的结果;
- 对于无法成功定位文件的请求，根据错误原因，作相应错误提示，并具备一定的异常情况处理能力。

1.3 系统设计

使用流式套接字进行编程，流式套接字是基于连接的编程，必须先建立连接后，才能从数据流中读出数据。

系统分为两个模块：

- server 服务器模块，负责创建套接字并监听，当连接客户端后通过新的线程将任务交给 sessionClient 模块处理响应
- sessionClient 处理请求模块，解析报文并回传响应

由流式套接字构造的系统流程如下所述：创建套接字，来用于监听；对监听套接字进行绑定；监听并等待连接；连接成功，生成会话套接字，用于接受客户端发来的数据流并且发送数据；对客户端的请求响应后通过会话套接字发送响应报文及文件；发送完之后关闭套接字。流程图 1.1 所示。

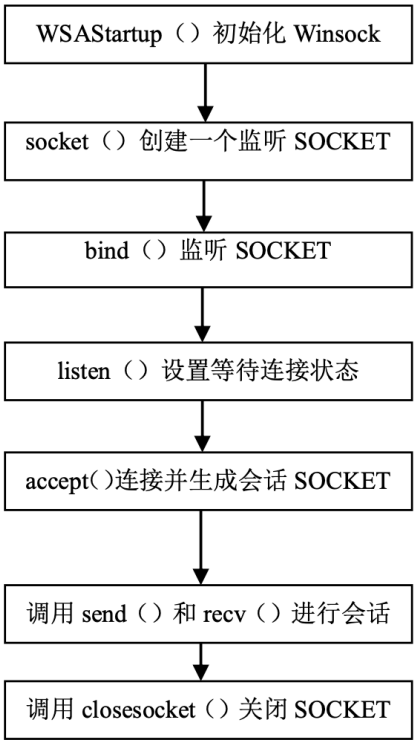

图 1.1: 服务器端套接字构造流程图

1.4 系统实现

1.4.1 server 服务器模块

1. 在 Windows 环境下首先需要初始化 winsock，使用 WSAStartup 函数；如果发生错误输出初始化失败信息并退出该程序，表示后续无法进行服务。

```
WSAData wsaData;
int nRc = WSAStartup(0x0202, &wsaData);
if (nRc) {
        cout<<"startup winsock error"<<endl;
        return 0;
}
```

2. 创建服务器监听套接字，使用数据类型 SOCKET，如果创建失败则输出创建失败信息。

```
SOCKET srvSocket = socket(AF_INET, SOCK_STREAM, 0);
if (srvSocket == INVALID_SOCKET)
        cout << "create socket error" << endl;
```

3. 配置 Web 服务器的监听地址、监听端口和主目录，由于要求这些设置不能写在代码中，不能每配置一次都要重编译代码，所以采用通过在控制台输入信息并读取的方式来进行设置。其中端口号和监听 IP 地址需要通过一个类型为 sockaddr_in 的地址变量绑定在套接字上，使用函数 bind，如果绑定失败，则输出监听绑定失败的信息并退出，否则无输出。

```
sockaddr_in addr;
int port;
char ip[20],content[20];
cout << "input ip" << endl;  cin >> ip;
cout << "input port" << endl;  cin >> port;
cout << "input content" << endl;  cin >> content;
fileposition = content;//fileposition为全局变量可以给sessionClient传递文件夹
    位置
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.S_un.S_addr = inet_addr(ip);
//binding
int rtn = bind(srvSocket, (LPSOCKADDR)&addr, sizeof(addr));
if (rtn == SOCKET_ERROR) {
        nSocketError = WSAGetLastError();
        cout << "bind error = " << nSocketError;
        return 0;
```

```
17  }
```

从控制台输入 IP 时，保存为点分十进制的 IP 地址的字符数组。但是套接字需要的 IP 地址为二进制无符号长整型，所以需要函数 inet_addr 将字符串形式的 IP 地址转换为 unsigned long 形式的值。输入端口号在内存中存储的是按电脑的小端存储的，但是网络是大端存储，故需要函数 htons 把主机字节顺序转换为网络字节顺序。

4. 此时套接字部分已经创建完成，需要将其配置为等待连接状态，也就是在没有有连接的时候保持监听状态，需要使用 listen 函数，如果设置失败则输出失败信息并退出，否则无输出。

```
1  rtn = listen(srvSocket, 5);
2  if (rtn == SOCKET_ERROR) {
3          nSocketError = WSAGetLastError();
4          cout << "listen error = " << nSocketError;
5          return 0;
6  }
```

5. 进入循环等待连接状态。首先生成会话套接字，需要通过 accept 函数连接并且得到客户端的 IP 地址和端口，如果连接失败则输出失败信息并返回，否则将该会话套接字通过一个新的线程交给 sessionClient 处理请求模块去处理。由于服务器端需要一直保持打开并监听的状态，等待连接的循环会一直运行。

```
1  while (true) {
2          int length = sizeof(sockaddr);
3          sockaddr_in clientaddr;
4          SOCKET sessionSocket = accept(srvSocket, (sockaddr *)&clientaddr, &
               length);
5          if (sessionSocket == INVALID_SOCKET) {
6                  nSocketError = WSAGetLastError();
7                  cout << "accept session error = " << nSocketError;
8                  return 0;
9          }
10         string str(content);
11         thread sessionThread(sessionToClient,sessionSocket);
12         std::cout << "request IP:" << (int)clientaddr.sin_addr.S_un.S_un_b.
               s_b1 << "." << (int)clientaddr.sin_addr.S_un.S_un_b.s_b2 << "."
               << (int)clientaddr.sin_addr.S_un.S_un_b.s_b3 << "." << (int)
               clientaddr.sin_addr.S_un.S_un_b.s_b4<<endl;
13         std::cout << "request port:" << clientaddr.sin_port << endl;
14         sessionThread.detach();
15 }
```

1.4.2 sessionClient 处理请求模块

1. 当有客户端的请求报文时，recv 函数便会从该套接字中接收并放置到指定的缓存区；

```
char* buf = (char*)malloc(1024 * sizeof(char));
int len = 1024;
int charnum = recv(sessionSocket, buf, len, 0);
if (charnum == SOCKET_ERROR) {
        charnum = WSAGetLastError();
        cout << "error" << endl;
        return;
}
```

2. 对接收到的消息进行解析，这里使用正则表达式进行匹配并获取请求报文中对应的信息。

```
const char* REQERROR = "HTTP/1.1 400 Bad Request\r\n";
//对接受到的消息进行解析
smatch strmatch;//正则表达式结果文本
regex regulation("([A-Za-z]+) +(.*) +(HTTP/[0-9][.][0-9])");//正则表达式规则
string str(buf);//需要用正则表达式的原始文本
int match = regex_search(str, strmatch, regulation);
if (match == 0) {//匹配成功返回1，失败返回0
        cout << "request message error"<<endl;
        sendre = send(sessionSocket, REQERROR, strlen(REQERROR), 0);
        closesocket(sessionSocket);
        return;
}
```

匹配成功后再解析得到请求方法和请求文件的 url。接着进一步进行正则匹配得到当前请求的文件类型。

```
map<string, string> content_type = {
        {".png", "image/png"},
        {".jpg", "image/jpeg"},
        {".jpeg", "image/jpeg"},
        {".html", "text/html"}
};
string msg_get = strmatch[1];//请求方法
string msg_url = strmatch[2];//请求文件的url
smatch filetype;
regex regulation2("\\..*");
match = regex_search(msg_url, filetype, regulation2);
if (match == 0) {
        std::cout << msg_get + msg_url + "\nFORMAT ERROR\n";
        sendre = send(sessionSocket, REQERROR, strlen(REQERROR), 0);
        closesocket(sessionSocket);
```

```
16          return;
17  }
18  string file_type = filetype[0];
```

根据文件夹地址和文件的 url 进行查找文件，如果没有找到文件则发送 NotFound 响应报文。

```
1  const char* NOTFOUND = "HTTP/1.1 404 Not Found\r\n";
2  ifstream f(fileposition + msg_url, std::ios::binary);
3  if (!f) {//没有找到文件
4          cout << msg_url + "\nNOT FOUND"<<endl;
5          sendre = send(sessionSocket, NOTFOUND, strlen(NOTFOUND), 0);
6          closesocket(sessionSocket);
7          return;
8  }
```

3. 读取文件中的内容并得到文件的 size，如果 size 小于等于 0 的话则响应请求错误。当 size 正常时，则按照文件内容发送响应报文。这里有一点就是加入长度超过了一个报文可以承载的容量时则分多个报文进行响应。

其中 HTTP 响应报文可以如下例子所示：

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

图 1.2: HTTP 响应报文例子

```
1  std::filebuf* tmp = f.rdbuf();
2  int size = tmp->pubseekoff(0, f.end, f.in);
3  tmp->pubseekpos(0, f.in);
4  if (size <= 0) {
5          sendre = send(sessionSocket, REQERROR, strlen(REQERROR), 0);
6          closesocket(sessionSocket);
7          return;
8  }else {
9          string Content_Type = content_type[file_type];
10         char* buffer = new char[size];
11         char* tail = buffer + size;
12         tmp->sgetn(buffer, size);
13         f.close();
```

```
14          cout << "successfully get file " + msg_url<<endl;
15          stringstream remsg;
16          remsg << "HTTP/1.1 200 OK\r\n" << "Connection:close\r\n"<<"Server:Zou
                Ya\r\n" << "Content Length:" << size << "\r\n" << "Content
                Type:" + Content_Type << "\r\n\r\n";
17          string remsgstr = remsg.str();
18          const char* remsgchar = remsgstr.c_str();
19          int tmpsize = strlen(remsgchar);
20          sendre = send(sessionSocket, remsgchar, tmpsize, 0);
21          while (buffer < tail) {
22                  sendre = send(sessionSocket, buffer, size, 0);
23                  buffer = buffer + sendre;
24                  size = size - sendre;
25          }
26          closesocket(sessionSocket);
27          return;
28  }
```

1.5 系统测试及结果说明

1. 配置 Web 服务器的监听地址、监听端口和主目录

图 1.3: 配置 Web 服务器

2. 能够开启多个线程处理请求报文。从连接套接字接收 http 请求报文，并根据请求报文的确定用户请求的网页文件；从服务器的文件系统中获得请求的文件。创建由请求的文件组成的 http 响应报文；经 TCP 连接向请求的客户端发送响应，并且浏览器可以正确显示网页的内容。

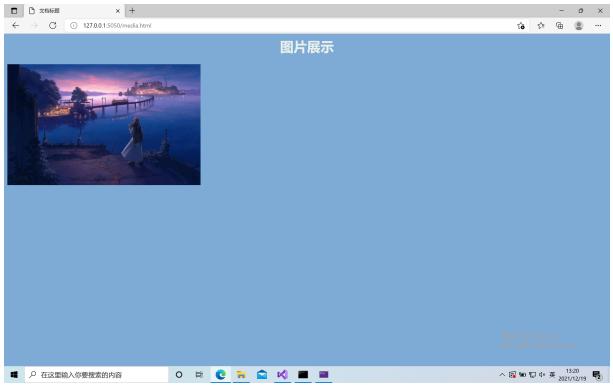

图 1.4: 浏览器显示网页

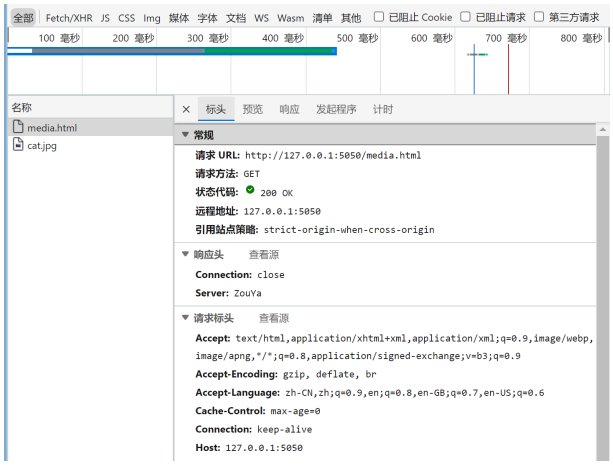

图 1.5: 浏览器请求和响应报文

3. 在服务器端的屏幕上输出请求的来源（IP 地址、端口号和 HTTP 请求命令行）

图 1.6: 控制台输出

4. 对于无法成功定位文件的请求，根据错误原因，作相应错误提示，并具备一定的异常情况处理能力。

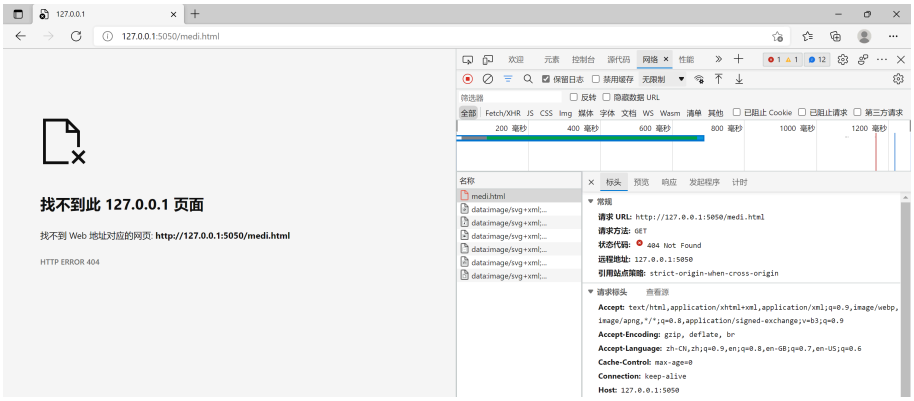

图 1.7: 无法找到网页时浏览器的显示

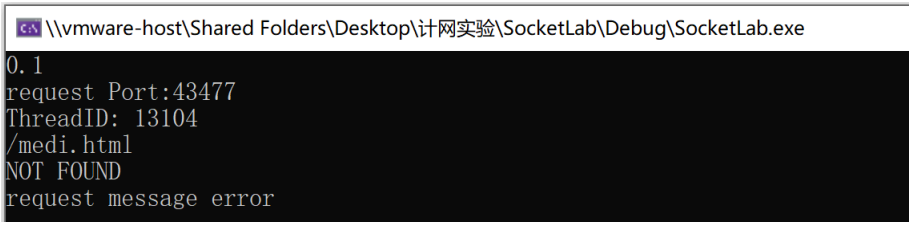

图 1.8: 无法找到网页时控制台的输出

1.6 其他需要说明的问题

开始做实验时，不知道怎么处理解析请求报文，后来再网上浏览找到了正则表达式的方法，这是一种十分简洁且方便的处理字符串的方法。

2 心得体会与建议

2.1 心得体会

在 Socket 编程实验中，通过自己编写代码真正实现服务器端处理客户端的 HTTP 请求，学到了很多，也和我们日常上网的体验相联系起来。

在上课的过程中，我们学习了关于 Socket 以及 HTTP 协议的理论知识，了解了理论知识在一定程度上也对网络请求以及响应有了了解，但是刚开始做实验的时候还是有一种很难着手去做的感觉。在明白了是以自己设备的文件系统作为服务器，并用自己的浏览器作为客户端去进行访问，而访问的 IP 就是本机 IP 之后，对整个实验有了一个框架性的理解，至少可以开始着手写了。

后来看了老师给的文档，对于 Windows 平台下网络通信协议的开发接口 Winsock 的 API 有了大致的了解，大概知道整个系统应该从哪开始和怎么构建。接下来难题变成了自己怎么去解析请求报文。一开始我打算自己根据 HTTP 请求头来进行解析，但是失败了。然后找到了正则表达式的解析方式，这就十分轻松简易了。

第二次和第三次的实验相比起来没有第一次那样无从下手的感觉。第二次的可靠传输协议在老师给的停等协议的基础上，理解了 GBN、SR 和 TCP 的原理就可以模仿着写出来了。第三次实验图形化的界面有一种做 Logisim 实验的感觉，把路由器、交换机等等硬件模拟出来，那些看不到的信息配置也通过图形化界面让我们可以直接输入接触到，感觉很有意思。

2.2 建议

在实验中都给了很详细的 API 说明文档，这对写代码很有帮助，这样我们就不用上网去找查 API 了。但是如果能把这个实验总体要达到的效果告诉我们，比如实验一中要用浏览器模拟客户端，我们对实验有一个大体的认识就不会产生无从下手的感觉了。

总体而言，像实验一和实验三这样贴近生活的实验，做起来还是很有意思的。

参考文献

[1] James F. Kurose, Keith W. Ross. 计算机网络: 自顶向下方法 (第 7 版) [M]. 机械工业出版社, 2018.