浙江大学

本科实验报告

课程名称：　计算机网络基础

实验名称：　基于 Socket 接口实现自定义协议通信

姓　　名：

学　　院：　计算机学院

系：　计算机科学与技术

专　　业：　计算机科学与技术

学　　号：

指导教师：　陆魁军

姓名	完成内容	百分比
	客户端+调试+实验报告	50%
	服务端+调试+实验报告	50%

2022 年 09　月　29　日

浙江大学实验报告

实验名称：基于 Socket 接口实现自定义协议通信　实验类型：编程实验

同组学生：＿＿＿＿＿＿＿＿实验地点：计算机网络实验室

一、 实验目的

- 学习如何设计网络应用协议
- 掌握 Socket 编程接口编写基本的网络应用软件

二、 实验内容

根据自定义的协议规范，使用 Socket 编程接口编写基本的网络应用软件。

- 掌握 C 语言形式的 Socket 编程接口用法，能够正确发送和接收网络数据包
- 开发一个客户端，实现人机交互界面和与服务器的通信
- 开发一个服务端，实现并发处理多个客户端的请求
- 程序界面不做要求，使用命令行或最简单的窗体即可
- 功能要求如下：
  1. 运输层协议采用 TCP
  2. 客户端采用交互菜单形式，用户可以选择以下功能：
     a) 连接：请求连接到指定地址和端口的服务端
     b) 断开连接：断开与服务端的连接
     c) 获取时间: 请求服务端给出当前时间
     d) 获取名字：请求服务端给出其机器的名称
     e) 活动连接列表：请求服务端给出当前连接的所有客户端信息（编号、IP 地址、端口等）
     f) 发消息：请求服务端把消息转发给对应编号的客户端，该客户端收到后显示在屏幕上
     g) 退出：断开连接并退出客户端程序
  3. 服务端接收到客户端请求后，根据客户端传过来的指令完成特定任务：
     a) 向客户端传送服务端所在机器的当前时间
     b) 向客户端传送服务端所在机器的名称
     c) 向客户端传送当前连接的所有客户端信息
     d) 将某客户端发送过来的内容转发给指定编号的其他客户端
     e) 采用异步多线程编程模式，正确处理多个客户端同时连接，同时发送消息的情况
- 根据上述功能要求，设计一个客户端和服务端之间的应用通信协议
- **本实验涉及到网络数据包发送部分不能使用任何的 Socket 封装类，只能使用最底层的 C 语言形式的 Socket API**
- 本实验可组成小组，服务端和客户端可由不同人来完成

三、 主要仪器设备

- 联网的 PC 机、Wireshark 软件
- Visual C++、gcc 等 C++集成开发环境。

四、 操作方法与实验步骤

- 设计请求、指示（服务器主动发给客户端的）、响应数据包的格式，至少要考虑如下问题：
  a) 定义两个数据包的边界如何识别
  b) 定义数据包的请求、指示、响应类型字段
  c) 定义数据包的长度字段或者结尾标记
  d) 定义数据包内数据字段的格式（特别是考虑客户端列表数据如何表达）
- 小组分工：1 人负责编写服务端，1 人负责编写客户端
- 客户端编写步骤（**需要采用多线程模式**）
  a) 运行初始化，调用 socket()，向操作系统申请 socket 句柄
  b) 编写一个菜单功能，列出 7 个选项
  c) 等待用户选择
  d) 根据用户选择，做出相应的动作（未连接时，只能选连接功能和退出功能）
    1. 选择连接功能：请用户输入服务器 IP 和端口，然后调用 connect()，等待返回结果并打印。连接成功后设置连接状态为已连接。**然后创建一个接收数据的子线程，循环调用 receive()，如果收到了一个完整的响应数据包，就通过线程间通信（如消息队列）发送给主线程，然后继续调用 receive()，直至收到主线程通知退出。**
    2. 选择断开功能：调用 close()，并设置连接状态为未连接。通知并等待子线程关闭。
    3. 选择获取时间功能：组装请求数据包，类型设置为时间请求，然后调用 send()将数据发送给服务器，**接着等待接收数据的子线程返回结果**，并根据响应数据包的内容，打印时间信息。
    4. 选择获取名字功能：组装请求数据包，类型设置为名字请求，然后调用 send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印名字信息。
    5. 选择获取客户端列表功能：组装请求数据包，类型设置为列表请求，然后调用 send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印客户端列表信息（编号、IP 地址、端口等）。
    6. 选择发送消息功能（选择前需要先获得客户端列表）：请用户输入客户端的列表编号和要发送的内容，然后组装请求数据包，类型设置为消息请求，然后调用 send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印消息发送结果（是否成功送达另一个客户端）。
    7. 选择退出功能：判断连接状态是否为已连接，是则先调用断开功能，然后再退出程序。否则，直接退出程序。
    8. 主线程除了在等待用户的输入外，还在处理子线程的消息队列，如果有消息到达，则进行处理，如果是响应消息，则打印响应消息的数据内容（比如时间、名字、客户端列表等）；如果是指示消息，则打印指示消息的内容（比如服务器转发的别的客户端的消息内容、发送者编号、IP 地址、端口等）。
- 服务端编写步骤（**需要采用多线程模式**）
  a) 运行初始化，调用 socket()，向操作系统申请 socket 句柄
  b) 调用 bind()，绑定监听端口（**请使用学号的后 4 位作为服务器的监听端口**），接着调用 listen()，设置连接等待队列长度
  c) 主线程循环调用 accept()，直到返回一个有效的 socket 句柄，在客户端列表中增加一个新客户端的项目，并记录下该客户端句柄和连接状态、端口。然后创建一个子线程后继续调用 accept()。该子线程的主要步骤是（**刚获得的句柄要传递给子线程，子线程内部要使用该句柄发送和接收数据**）：

- ✧ 调用 send()，发送一个 hello 消息给客户端（可选）
- ✧ 循环调用 receive()，如果收到了一个完整的请求数据包，根据请求类型做相应的动作：
  1. 请求类型为获取时间：调用 time()获取本地时间，然后将时间数据组装进响应数据包，调用 send()发给客户端
  2. 请求类型为获取名字：将服务器的名字组装进响应数据包，调用 send()发给客户端
  3. 请求类型为获取客户端列表：读取客户端列表数据，将编号、IP 地址、端口等数据组装进响应数据包，调用 send()发给客户端
  4. 请求类型为发送消息：根据编号读取客户端列表数据，如果编号不存在，将错误代码和出错描述信息组装进响应数据包，调用 send()发回源客户端；如果编号存在并且状态是已连接，则将要转发的消息组装进指示数据包。调用 send()发给接收客户端（使用接收客户端的 socket 句柄），发送成功后组装转发成功的响应数据包，调用 send()发回源客户端。

d) 主线程还负责检测退出指令（如用户按退出键或者收到退出信号），检测到后即通知并等待各子线程退出。最后关闭 Socket，主程序退出。

- 编程结束后，双方程序运行，检查是否实现功能要求，如果有问题，查找原因，并修改，直至满足功能要求
- 使用多个客户端同时连接服务端，检查并发性
- 使用 Wireshark 抓取每个功能的交互数据包

五、 实验数据记录和处理

请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **源代码：客户端和服务端的代码分别在一个目录**
- **可执行文件：可运行的.exe 文件或 Linux 可执行文件，客户端和服务端各一个**

- 描述请求数据包的格式，请求类型的定义


```
structure Package{
    char type[4];
    char message[MAX_MESSAGE_LENGTH];
}
```

我们在编写之初对于数据包的定义是如上图所示，不过后来发现不同请求差别过大而且不需要太整齐的格式就足以完成要求，所以主要是采用字符串比较的方式完成的实验。下面我将逐一介绍客户端的各种请求命令。在客户端用户共有 8 种输入格式，客户端根据这八种字符串判断应该执行的动作。除了“:h”是用来展示帮助界面的之外，其他七种输入都对应一种请求数据包。

1. “:c”，直接输入:c 后调用 ConnectFunc()函数，函数中首先调用 socket 函数创建客户端的 socketfd，然后根据提示信息依次输入服务器端 ip 地址以及监听端口，并使用 connect 函数进行连接。

2. “:d” ，客户端输入:d 后首先在本地判断是否有建立连接，如果未建立连接则输出提示信息“Failed. You have not connected!”，如果建立了连接，则直接向服务器发送字符‘d’请求断开连接。

3. “:t” ，客户端输入:t 同样在本地判断是否有建立连接，如果未建立连接则输出提示信息，如果建立了连接，则直接向服务器发送字符‘t’请求服务器时间。

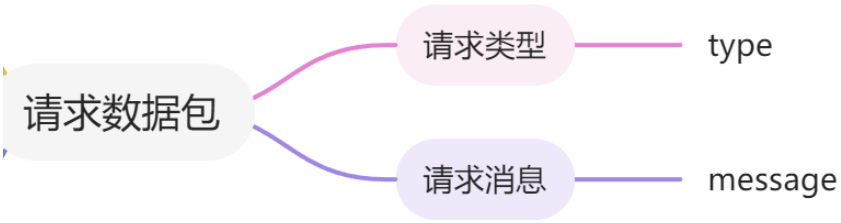
4. “:n” ，客户端输入:n 同样在本地判断是否有建立连接，如果未建立连接则输出提示信息，如果建立了连接，则直接向服务器发送字符‘n’请求服务器名称。

5. “:l” ，客户端输入:l 同样在本地判断是否有建立连接，如果未建立连接则输出提示信息，如果建立了连接，则直接向服务器发送字符‘l’请求服务器发送客户端列表信息。

6.“:s” ，客户端输入:s 同样在本地判断是否有建立连接，如果未建立连接则输出提示信息，如果建立了连接，则根据提示信息依次输入目标客户端编号和想要发送的内容。客户端将这些内容拼接起来后向服务器发送字符串“s 目标客户端编号 发送内容”，中间用空格隔开。

7.“:e” ，客户端输入:e 同样在本地判断是否有建立连接，如果未建立连接则直接退出客户端程序，如果建立了连接，则向服务器发送字符‘d’请求断开连接，完成断开连接之后再退出程序。

- 描述响应数据包的格式，响应类型的定义




```
structure Package{
    char type[4];
    char message[MAX_MESSAGE_LENGTH];
}
```

服务器对于客户端的反馈类型由开头的两个字符判断

1. 对于建立连接请求，这和其他请求不同，服务器使用 accept 函数接受，连接成功则发送"hello!"字符串
2. 对于断开连接请求，发送“:d”说明断开成功
3. 对于请求时间，发送“:t 时间”，中间使用空格隔开
4. 对于请求名称，发送“:n 服务器名称”，中间使用空格隔开
5. 对于请求客户端列表请求，发送“:l ID NAME IP PORT”，如果有多个客户端则不断重复 ID NAME IP PORT，最终组合为一个长字符串发送过去
6. 对于发送信息请求，直接向目标客户端发送信息，以:s 开头，例如“:s hahah”，如果发送成功，则向源客户端发送“:s success”，如果发送失败则向源客户端发送“:s fail”

这些信息的格式在客户端都已经和服务器统一好了，只需根据前两个字符判断就可以正确获取信息。

- 客户端初始运行后显示的菜单选项

  共有 8 条命令可供选择

```
Please input a command:
+--------------------------------+--------------------------------+
|            Command             |            Function            |
+--------------------------------+--------------------------------+
|   :c                           |      Connect to a server.      |
|   :d                           |      Disconnect.               |
|   :t                           |      Get the server time.      |
|   :n                           |      Get the server name.      |
|   :l                           |      Get the client list.      |
|   :s [CLIENT NUM] [MESSAGE]    |      Send a message.           |
|   :e                           |      Exit.                     |
|   :h                           |      Get help info.            |
+--------------------------------+--------------------------------+

[INFO] Waiting for new command...
```

- 客户端的主线程循环关键代码截图（描述总体，省略细节部分）

  程序根据用户输入前两个字符判断输入是否合法以及是哪条命令

```c
while(ret){
    scanf("%s", cmd);
    switch (cmd[1])
        {
        case 'c':// 连接
            ConnectFunc(); break;// 正常建立连接
        case 'd': // 断开连接
            Disconnect_Func(sockfd);break;
        case 't': // 获取时间
            send(sockfd, "t", 1, 0);break;
        case 'n': // 获取服务器name
            send(sockfd, "n", 1, 0);break;
        case 'l': // 获取所有已连接客户端信息
            send(sockfd, "l", 1, 0);break;
        case 's': // 输入要发送的信息
            send(sockfd, buffer, strlen(buffer), 0);break;
        case 'e':
            Disconnect_Func(sockfd);
            ret = 0;break;
        case 'h':
            Help();break;
        default:break;
```

- 客户端的接收数据子线程循环关键代码截图（描述总体，省略细节部分）

  客户端根据前两个字符判断接受信息类型，然后分别处理

```
while(1) { //接收循环
    if(recv(socket_h, buffer, MAX_DATA, 0) > 0) {//处理服务器发来的包
    while(strlen(ptr) && ptr[0] == ':') {
    // 收到服务器发来的信息并输出；每一次请求的返回结果用:t/:n/:l/:s区分
    switch (ptr[1]) {
        case 't': // 获取时间 输出序列 :t time:t time
            cout << "Server time: " << endl << rec << endl << endl;
            break;
        case 'n': //获取服务端主机名 :n servername
            cout << "Server name: " << endl << rec << endl << endl;
            break;
        case 'l': // 获取客户端列表 :l list
            cout << "client list: " << endl;
            printf("%4s%10s%15s%10s\n", "ID", "NAME", "IP", "PORT");
            break;
        case 's': // 获取信息 :s message
            cout << "You received a message: " << endl << rec ;
        case 'd': // 同意断开连接 :d
            pthread_exit(NULL); // 结束该线程
            break;
        default:
            break;
}}}}
```

- 服务器初始运行后显示的界面

  服务器端根据提示信息自主选择开放的端口

```
rain@rain-virtual-machine:~/jw_lab/lab5$ ./myserver
[INFO] Which port do you want to open?
6666

[INFO] Open socket successfully.
[INFO] Bind socket successfully.
[INFO] Server starts.
[INFO] Server is listening...
```

- 服务器的主线程循环关键代码截图（描述总体，省略细节部分）

  服务器每接受成功一个客户端就新创建一个线程处理

```
while(1) {
    // 接受客户端的连接
    clientfd = accept(serverfd, (struct sockaddr*)&clientaddr, (socklen_t*)&socklen);
    if(clientfd == -1)
        perror("accept");
    else { // 连接成功，增加用户
        //检验名字
        if (iret = recv(clientfd, clientName, sizeof(clientName), 0) <= 0)
            perror("name");
        //设置客户端信息
        client[clientNum].clientfd = clientfd;                          // 客户端描述符
        strcpy(client[clientNum].name, clientName);                     // 客户端主机名
        strcpy(client[clientNum].ip, inet_ntoa(clientaddr.sin_addr));   // 客户端ip地址
        client[clientNum].port = clientaddr.sin_port;                   // 客户端端口号
        //打印当前连接用户的信息
        printClient();
        //创建新线程
        pthread_t pid;
        send(clientfd, "hello!", 6, 0); // 发送hello信息给客户端
        pthread_create(&pid, NULL, threadWork, &clientfd);
    }
}
```

- 服务器的客户端处理子线程循环关键代码截图（描述总体，省略细节部分）

  同样服务器根据发送数据包的第一个字符判断是什么命令

```
while(1) {
        memset(buffer, 0, sizeof(buffer));
        if (iret = recv(clientfd, buffer, sizeof(buffer), 0) > 0) {
            switch(buffer[0]) {                                         // 成功收到消息
                case 's': // 发送数据，例如"s 3 Hello world"
                    temp=StrTok();                                      // 按照分隔符切分发送数据
                    send(client[dstfd - 1].clientfd, temp, strlen(temp), 0);
                    break;                                              // 向客户端发送数据
                case 't': // 获取时间
                    sprintf(buffer, ":t %d-%d-%d %s %d:%d:%d", \// 取得服务器当前时间
                        (1900 + p->tm_year), (1 + p->tm_mon), p->tm_mday, \
                        wday[p->tm_wday], p->tm_hour, p->tm_min, p->tm_sec);
                    send(clientfd, buffer, strlen(buffer), 0);  // 向客户端发送响应结果
                    break;
                case 'l': // 获取所有客户端信息
                    buffer=ClientList();                                // 获取所有客户端信息
                    send(clientfd, buffer, strlen(buffer), 0);  // 向客户端发送响应结果
                    break;
                case 'n': // 获取用户名
                    gethostname(buffer + 3, MAX_NAME_LENGTH);   // 获得当前主机名
                    send(clientfd, buffer, strlen(buffer), 0);  // 向客户端发送响应结果
                    break;
                case 'd': // 断开连接
                    disconnect(clientfd);
                    break;
                default:
                    break;
            }
        }
    }
    printf("[INFO] Thread[%lu] ends\n\n", pthread_self());
    pthread_exit(NULL); // 销毁线程
    return NULL;
}
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

  客户端：

  客户端根据提示信息输入服务器 ip 地址、开放端口以及名称

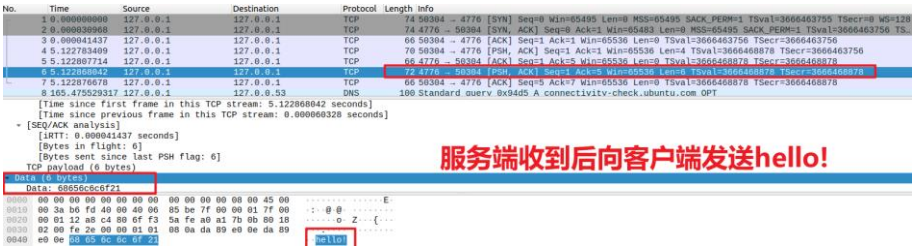
```
[INFO] Waiting for new command...
:c
[INFO] Please input server ip: 127.0.0.1
[INFO] Please input server port: 6666

[INFO] Client has connected to server(127.0.0.1:6666)!
[INFO] Please input a username:  [no longer than 20]
zhm
[INFO] Received contents from server: hello!

[INFO] Waiting for new command...
```

服务端：

连接成功后显示客户端 ip 地址，端口以及名称

```
[INFO] New client has connected.
Client IP: 127.0.0.1
Client Port: 16520
Client Name: zhm
Client FD: 4
```

Wireshark 抓取的数据包截图：



- 客户端选择获取时间功能时，客户端和服务端显示内容截图。

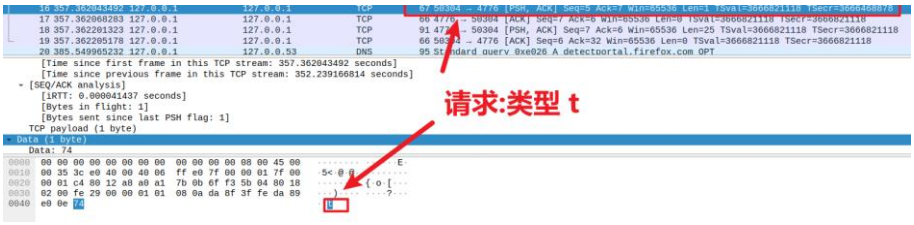
  客户端：

```
[INFO] Waiting for new command...
:t
[INFO] Received message.
Server time:
2022-10-4 Tue 21:40:53

[INFO] Waiting for new command...
```

服务端：

```
[INFO] Information received from client(4):t
[INFO] Sending server time to other client...
Current server time: :t 2022-10-4 Tue 21:40:53
Send message successfully
```

Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的时间数据对应的位置）：



- 客户端选择获取名字功能时，客户端和服务端显示内容截图。

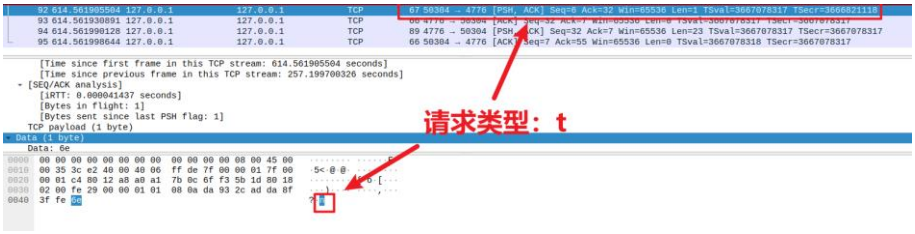
  客户端显示内容：

```
:n
[INFO] Received message.
Server name:
rain-virtual-machine

[INFO] Waiting for new command...
```

服务端显示内容：

```
[INFO] Information received from client(5):n
[INFO] Sending server name to a client...
server name: rain-virtual-machine
Send message successfully
```

Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的名字数据对应的位置）：



相关的服务器的处理代码片段：

```c
case 'n': // 获取用户名
    printf("[INFO] Sending server name to a client...\n");
    memset(buffer, 0, sizeof(buffer));
    strcpy(buffer, ":n ");
    gethostname(buffer + 3, MAX_NAME_LENGTH); // 获得当前主机名
    printf("server name: %s\n", buffer + 3);
    if((iret = send(clientfd, buffer, strlen(buffer), 0)) <= 0) // 向客户端发送响应结果
        perror("n send");
    printf("Send message successfully\n\n");
    break;
```

- 客户端选择获取客户端列表功能时，客户端和服务端显示内容截图。

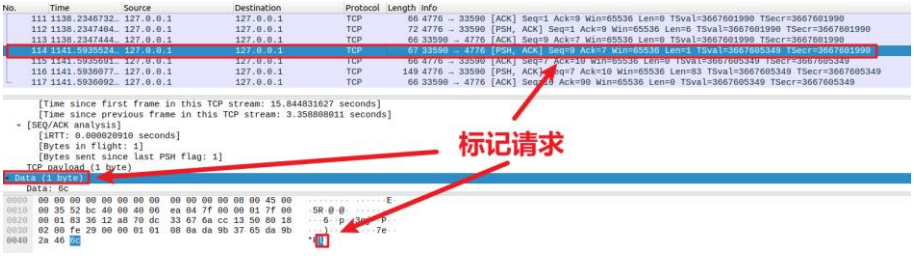
  客户端显示内容：

```
[INFO] Waiting for new command...
:l
[INFO] Received message.
client list:
  ID       NAME              IP       PORT
   1       rain       127.0.0.1      32964
   2  xiaoting        127.0.0.1      13955

[INFO] Waiting for new command...
```

服务端显示内容：

```
[INFO] Information received from client(4):l
[INFO] Sending client list to a client...
  ID       NAME              IP       PORT
   1       rain       127.0.0.1      32964
   2  xiaoting        127.0.0.1      13955
Send message successfully
```

Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的客户端列表数据对应的位置）：



相关的服务器的处理代码片段：

```c
case '1': { // 获取所有客户端信息， 每个客户端以回车结尾，客户端每个字段以空格区分
    printf("[INFO] Sending client list to a client...\n");
    char temp[MAX_NAME_LENGTH];
    memset(temp, 0, sizeof(temp));
    memset(buffer, 0, sizeof(buffer));
    strcpy(buffer, ":1 ");
    printf("%4s%10s%15s%10s\n", "ID", "NAME", "IP", "PORT");
    for(int i = 0; i < clientNum; i++) {
        sprintf(temp, "%4d%10s%15s%10d\n", i + 1, client[i].name, client[i].ip, client[i].port);
        printf("%s", temp);
        strcat(buffer, temp);
        memset(temp, 0, sizeof(temp));
    }
    if((iret = send(clientfd, buffer, strlen(buffer), 0)) <= 0) // 向客户端发送响应结果
        perror("1 send");
    printf("Send message successfully\n\n");
    break;
}
```

- 客户端选择发送消息功能时，客户端和服务端显示内容截图。

发送消息的客户端：

```
[INFO] Waiting for new command...
:s
Please input client ID:
1
Please input the message:
HELLO!!!

[INFO] Received message.
You received a message:
success

[INFO] Waiting for new command...
```

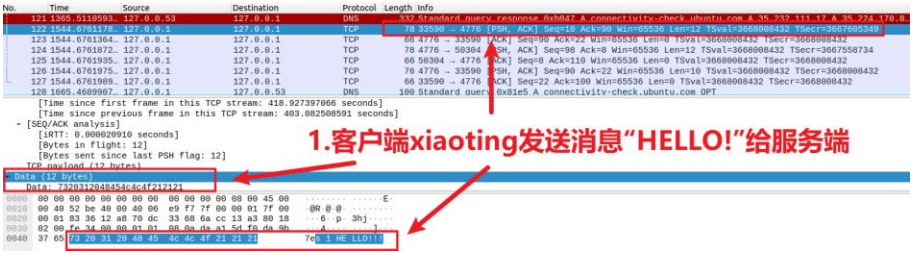
服务器：

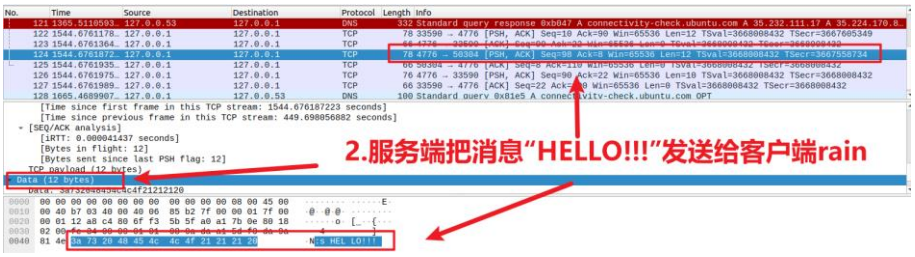
```
[INFO] Information received from client(4):s 1 HELLO!!!
[INFO] Sending message to other client...
Send message to target client[5] successfully
Send successful message to source client[4]
```

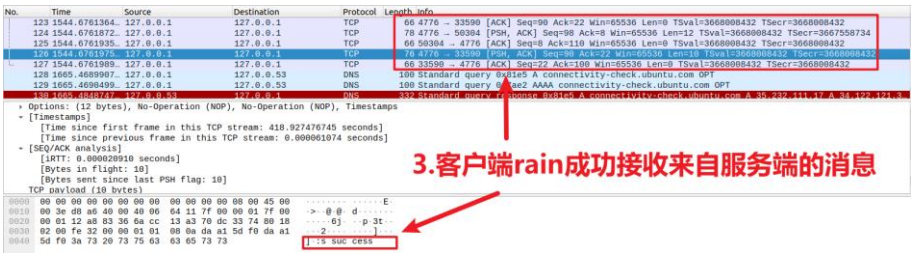
接收消息的客户端：

```
[INFO] Received message.
You received a message:
HELLO!!!

[INFO] Waiting for new command...
```

Wireshark 抓取的数据包截图（发送和接收分别标记）：







相关的服务器的处理代码片段：

```
case 's': { // 发送数据，例如"s 3 Hello world"
    printf("[INFO] Sending message to other client...\n");
    int iret;
    int count = 0;
    int dstfd = 0; // 目标客户端编号
    char temp[MAX_TEXT_LENGTH]; // 存储传输内容
    memset(temp, 0, sizeof(temp));
    strcpy(temp, ":s ");
    const char* delim = " "; // 空格作为分隔符切分字符串
    char* separate_str = NULL; // 存储切分的字符串
    separate_str = strtok(buffer, delim); // 按照分隔符切分
    while(separate_str != NULL) {
        if (count == 0); // s
        else if (count == 1) // 12, 获得文件描述符
            dstfd = atoi((const char*)(separate_str));
        else {
            strcat(temp, separate_str);
            strcat(temp, " "); // 多分一个问题不大
        }
        separate_str = strtok(NULL, delim); // 后续调用
        count++;
    }
    if(dstfd > clientNum) { // 传输失败，发送错误信息给源客户端
        send(clientfd, ":s fail", 7, 0);
        printf("Send error message to source client[%d]\n\n", clientfd);
    }
    else {
        if ((iret = send(client[dstfd - 1].clientfd, temp, strlen(temp), 0)) == -1) {
            send(clientfd, ":s fail", 7, 0);
            printf("Send error message to source client[%d]\n\n", clientfd);
        }
        else {
            send(clientfd, ":s success", 10, 0); // 传输成功，发送成功信息
            printf("Send message to target client[%d] successfully\n", client[dstfd - 1].clientfd);
            printf("Send successful message to source client[%d]\n\n", clientfd);
        }
    }
    break;
}
```

相关的客户端（发送和接收消息）处理代码片段：

发送消息：

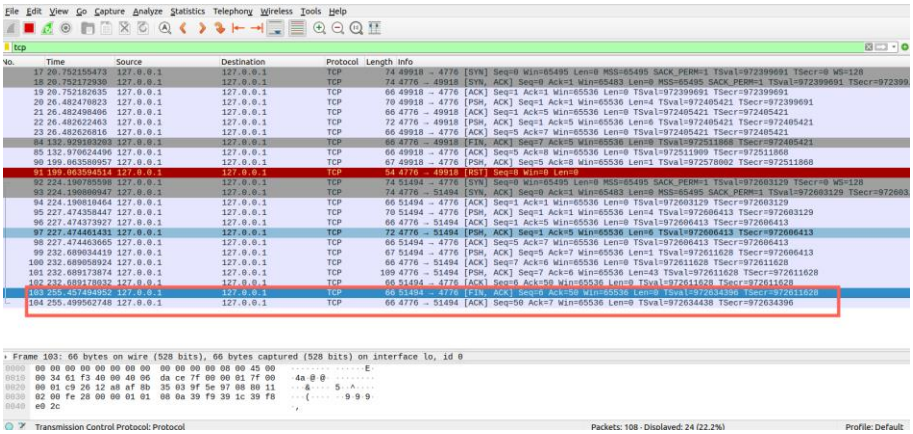
```
case 's': // 输入要发送的信息
    if (isConnected) {
        char buffer[MAX_DATA];
        memset(buffer, 0, sizeof(buffer));
        char des[12];
        char mes[MAX_DATA - 14];
        cout << "Please input client ID:" << endl;
        cin >> des;
        cout << "Please input the message:" << endl;
        cin >> mes;
        // getchar();
        cout << endl;
        //信息整合到buffer中，并发给服务端
        sprintf(buffer,"s ");
        strcat(buffer, des);
        strcat(buffer, " ");
        strcat(buffer, mes);
        send(sockfd, buffer, strlen(buffer), 0);
    }
    else {
        cout << "Failed. You have not connected!" << endl << endl;
        cout << "[INFO] Waiting for new command..." << endl;
    }
```

接收消息：

```
while(1) { //接收循环
    char buffer[MAX_DATA] = {0};
    if(recv(socket_h, buffer, MAX_DATA, 0) > 0) {
        //处理服务器发来的包
        printf("[INFO] Received message.\n");
        char* ptr = buffer;
        char rec[MAX_DATA];
        memset(rec, 0, MAX_DATA);
        while(strlen(ptr) && ptr[0] == ':') {
            // 收到服务器发来的信息并输出；每一次请求的返回结果用:t/:n/:l/:s区分
            for(i = 3; i < strlen(ptr); i++)
                rec[i - 3] = ptr[i];
            switch (ptr[1]) {
            case 's': // 获取信息 :s message
                cout << "You received a message: " << endl << rec << endl << endl;
                break;
```

- 拔掉客户端的网线，然后退出客户端程序。观察客户端的 TCP 连接状态，并使用 Wireshark 观察客户端是否发出了 TCP 连接释放的消息。同时观察服务端的 TCP 连接状态在较长时间内（10 分钟以上）是否发生变化。

在这里我们直接按 Ctrl-C 退出客户端程序，观察到 wireshark 状态如下所示。用红框标出的信息为这一过程时服务器和客户端互相发送的信息。客户端被动关闭后发出 FIN，服务器端则发出 ACK 回应。这说明客户端发出了 TCP 连接释放请求，服务端的 TCP 连接再较长时间内不发生改变，始终为 ESTABLISHED 状态。



使用 netstat 命令也可以观察到 Ctrl-C 前后状态的改变

```
rain@rain-virtual-machine:~$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:4776          localhost:51494         CLOSE_WAIT
tcp        0      0 localhost:45544         localhost:4776          ESTABLISHED
tcp        0      1 rain-virtual-mach:39822 17.111.232.35.bc.g:http SYN_SENT
tcp        0      0 localhost:4776          localhost:45544         ESTABLISHED
rain@rain-virtual-machine:~$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:4776          localhost:51494         CLOSE_WAIT
tcp        0      0 rain-virtual-mach:56060 17.111.232.35.bc.g:http ESTABLISHED
tcp        0      0 localhost:45544         localhost:4776          FIN_WAIT2
tcp        0      1 rain-virtual-mach:41914 32.121.122.34.bc.g:http SYN_SENT
tcp        0      0 localhost:4776          localhost:45544         CLOSE_WAIT
tcp        0      0 localhost:4776          localhost:45544         CLOSE_WAIT
```

- 再次连上客户端的网线，重新运行客户端程序。选择连接功能，连上后选择获取客户端列表功能，查看之前异常退出的连接是否还在。选择给这个之前异常退出的客户端连接发送消息，出

现了什么情况？

之前连接的客户端的会话过程如下所示，连接成功后使用 Ctrl-C 中断程序

```
[INFO] Waiting for new command...
:c
[INFO] Please input server ip: 127.0.0.1
[INFO] Please input server port: 2635

[INFO] Client has connected to server(127.0.0.1:2635)!
[INFO] Please input a username:  [no longer than 20]
c1
[INFO] Received contents from server: hello!

[INFO] Waiting for new command...
:l
[INFO] Received message.
client list:
  ID      NAME             IP      PORT
   1        c1      127.0.0.1     27871

[INFO] Waiting for new command...
^C
```

随后重新创建一个客户端，获取客户端列表功能时发现之前异常退出的连接还在

```
[INFO] Waiting for new command...
:l
[INFO] Received message.
client list:
  ID      NAME             IP      PORT
   1        c1      127.0.0.1     27871
   2        c2      127.0.0.1     28383
```

如果重新发消息的话在新客户端显示发送成功，但因为旧客户端程序已经退出，所以无法收到刚发出的消息

```
[INFO] Waiting for new command...
:s
Please input client ID:
1
Please input the message:
hahah

[INFO] Received message.
You received a message:
success
```

- 修改获取时间功能，改为用户选择 1 次，程序内自动发送 100 次请求。服务器是否正常处理了 100 次请求，截取客户端收到的响应（通过程序计数一下是否有 100 个响应回来），并使用 Wireshark 抓取数据包，观察实际发出的数据包个数。

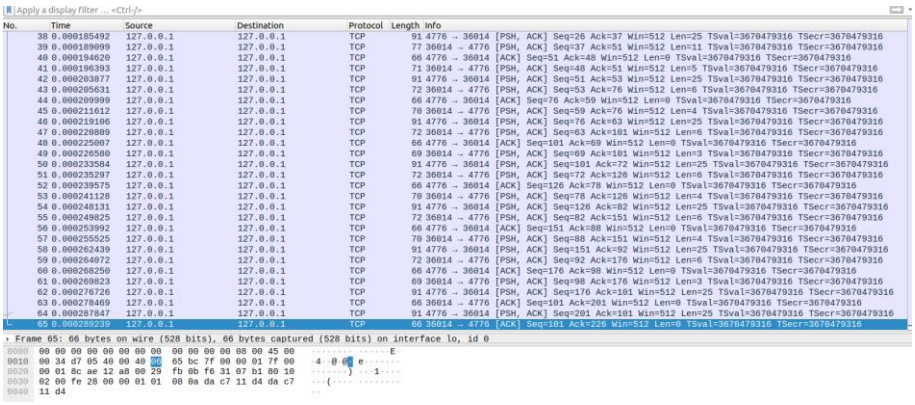
  服务器没有正常处理 100 次请求

  客户端收到的响应 9 次

  wireshark 统计实际发出的数据包个数 65 个包，而一次正常的通信需要往返共 4 个包

```
Server time:
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19:t
2022-10-5 Wed 16:53:19

[INFO] Waiting for new co
mmand...
```

```
[INFO] Information received from client(4):t
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttttttttttttttttttttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully

[INFO] Information received from client(4):ttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 16:53:19
Send message successfully
```



- 多个客户端同时连接服务器，同时发送时间请求（程序内自动连续调用 100 次 send），服务器和客户端的运行截图

服务器运行截图：

```
[INFO] Information received from client(5):t
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttttttttttttttttttttttttttttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):ttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(5):tttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:2
Send message successfully

[INFO] Information received from client(4):ttttttttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:3
Send message successfully

[INFO] Information received from client(4):ttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
[INFO] Sending server time to other client...
Current server time: :t 2022-10-5 Wed 17:7:3
Send message successfully
```

客户端 1 的运行截图：

```
received times:1
Server time:
2022-10-5 Wed 17:7:2:t 2022-10-5 Wed 17:7:2:t
2022-10-5 Wed 17:7:2:t 2022-10-5 Wed 17:7:2:t
2022-10-5 Wed 17:7:2:t 2022-10-5 Wed 17:7:2:t
2022-10-5 Wed 17:7:2

[INFO] Waiting for new command...
[INFO] Received message.
received times:2
Server time:
2022-10-5 Wed 17:7:2:t 2022-10-5 Wed 17:7:2

[INFO] Waiting for new command...
```

客户端 2 的运行截图：

```
[INFO] Received message.
received times:1
Server time:
2022-10-5 Wed 17:7:3:t 2022-10-5 Wed 17:7:3

[INFO] Waiting for new command...
```

说明，在连续快速的发送请求时，会造成数据包的合并，相当于本来应该分几次发的数据包合并成了一个，而服务器对命令的判断显然判定为无效命令（如本该是”t”命令，结果接收到”ttttttttttttt”，则无法正确识别

六、 实验结果与分析

- 客户端是否需要调用 bind 操作？它的源端口是如何产生的？每一次调用 connect 时客户端的端口是否都保持不变？

  客户端不需要 bind 操作

  它的源端口由 socket 随机创建

  每一次客户端的端口会发生变化，因为客户通过哪个端口与服务器建立连接并不重要，socket 执行体会为程序自动选择一个未被占用的端口

- 假设在服务端调用 listen 和调用 accept 之间设了一个调试断点，暂停在此断点时，此时客户端调用 connect 后是否马上能连接成功？

  客户端调用 connect 后可以马上连接成功。

- 连续快速 send 多次数据后，通过 Wireshark 抓包看到的发送的 Tcp Segment 次数是否和 send 的次数完全一致？

  通过 Wireshark 抓包看到的发送的 Tcp Segment 次数和 send 的次数不一定完全一致。TCP 传输承载的最大长度是 1200 字节，如果超出 1200 字节，则发送的数据包会被拆成多个数据包发送。

- 服务器在同一个端口接收多个客户端的数据，如何能区分数据包是属于哪个客户端的？

  本实验中服务端把所有连接状态的客户端信息都宝存在了 client[BACKLOG]结构体中,根据 IP,port,name,clientfd 都可以区分数据包是属于哪个客户端

```
struct connect{
    char name[MAX_NAME_LENGTH]; // 客户端名称
    int clientfd;               // 客户端描述符
    char ip[MAX_NAME_LENGTH];   // 客户端 ip 地址
    int port;                   // 客户端端口
}client[BACKLOG];
```

- 客户端主动断开连接后，当时的 TCP 连接状态是什么？这个状态保持了多久？（可

以使用 netstat -an 查看）

在服务器端与客户端连接时 TCP 的连接状态是 ESTABLISHED

```
tcp        0      0 127.0.0.1:2635          127.0.0.1:57174         ESTABLISHED
```

当客户端发送’d’命令主动断开连接后 TCP 状态马上变为 FIN_WAIT2

```
tcp        0      0 127.0.0.1:2635          127.0.0.1:57174         FIN_WAIT2
```

大约一分钟之后这条连接消失

- 客户端断网后异常退出，服务器的 TCP 连接状态有什么变化吗？服务器该如何检测连接是否继续有效？

  服务器的 TCP 连接始终为 ESTABLISHED 状态，没有改变。查阅资料后发现可以采用“心跳机制”解决客户端异常退出的问题。客户端定时向服务端发送空消息（ping），服务端启动心跳检测，超过一定时间范围没有新的消息进来就默认为客户端已断线，服务端主动执行 close()方法断开连接

七、 讨论、心得

客户端：

编写难点主要在于这是第一个实验，对于 socket 编程、wireshark 使用方法、配置环境不大清楚，通过查询大量资料后解决了问题。在测试环节中，因为我的 wsl 不能正常使用，所以在 ubuntu22.04 中安装了 wireshark,通过设置过滤器为 lookbackinto:lo，成功捕获了本地数据包

服务端：

这个实验还是很有意义的，我学习了 Linux 下的网络编程以及 C/C++多线程的使用方法，收获很大。由于在编写过程中由于对网络原理不是很了解，所以在编写、调试、测试的过程中都遇到了不少原理上的障碍。比如说测试时对 TCP 的各种状态没有什么概念，wireshark 和 netstat 命令展示的结果不清楚具体是什么意思，在查阅了大量资料之后才顺利完成，所以颇费了一番周折。

关于实验安排的建议是：我认为这个实验应该安排在理论知识学习完成之后再做。