

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： 朱懿

学 院： 工程师学院

系：

专 业： 计算机技术

学 号： 21960441

指导教师：

2019 年 10 月 16 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： 无 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（画图说明），请求类型的定义

类型字段	数据字段	结束字段（\$@）
------	------	-----------

类型字段：

总共五种类型

T：表示请求时间

N：表示请求名字

L：表示请求客户端列表

M：表示发送数据给另一个客户端

C：向服务器发送消息表示本客户端要关闭（正常关闭情形）

数据字段：

根据实际需求，通常情况下，只有 M 类型的请求消息需要携带数据字段，其他类型的消息则隐藏该字段

结束字段：

用\$@符号表示该报文结束

● 描述响应数据包的格式（画图说明），响应类型的定义

类型字段	ID 字段	数据字段	结束字段（\$@）
------	-------	------	-----------

类型字段：

同上述请求数据包的类型字段

ID 字段：

向发送申请的客户端显示该客户端在服务器列表中的 id

其格式为 “Your ID: ” +实际 id+ “\n”

数据字段：

向请求的客户端显示请求内容

结束字段：

用\$@符号表示该报文结束

● 描述指示数据包的格式（画图说明），指示类型的定义

类型字段（M）	来源 ID 字段	数据字段	结束字段（\$@）
---------	----------	------	-----------

类型字段：

因为只有向另一个客户端转发需要这个指示数据包，所以就一个”M”了

来源 ID 字段：

需要指出是哪个客户端向本客户端发送的

其格式为 “Source ID: ” +来源 id + “\n”

数据字段：

发送一段消息

结束字段

用\$@符号表示该报文结束

● 客户端初始运行后显示的菜单选项

```
E:\Graduate\2019-2020First\ComputerNetworks\Experiments\Experiment7\Client\Client\Debug\Client.exe
*-----system-----*
Start Client
#####
#      Thank you for using this system      #
#      please choose from the following choices      #
#      1:connect      #
#      2:close      #
#      3:get time info      #
#      4:get name info      #
#      5:get client list      #
#      6:sent message      #
#      7:exit      #
#####
*-----system-----*
please enter your choose:
_
```

如图

给出一个界面显示七个选项

- 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）

Emmm 有点长

```

choose = Screen(true);
while (true) {
    switch (choose) {
    case 1:
        if (state == 1) {
            printf("*-----system-----*\n");
            printf("This Client has been connected\n");
            break;
        }

        sListen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (sListen == INVALID_SOCKET)
        {
            WSACleanup();
            printf("*-----system-----*\n");
            printf("socket() failed!\n");
            return -1;
        }

        printf("*-----system-----*\n");
        printf("Please enter the target IP:\n");
        scanf("%s", &IP);
        printf("*-----system-----*\n");
        printf("Please enter the target Port:\n");
        scanf("%d", &port);
        sServer.sin_family = AF_INET;
        sServer.sin_port = htons(port);
        sServer.sin_addr.S_un.S_addr = inet_addr(IP);

        connectRet = Connect();
        if (connectRet == -1)
            return -1;
        break;
    case 2:
        if (state == 1)
            Close();
        break;
    case 3:
        if (state == 0) {
            printf("*-----system-----*\n");
            printf("Please connect first\n");
        }
        else
        {
            Ask('T');
            lock_print(1);
        }
        break;
    }
}

```

```

case 4:
    if (state == 0) {
        printf("*-----system-----*\n");
        printf("Please connect first\n");
    }
    else
    {
        Ask('N');
        lock_print(2);
    }
    break;
case 5:
    if (state == 0) {
        printf("*-----system-----*\n");
        printf("Please connect first\n");
    }
    else
    {
        Ask('L');
        lock_print(3);
        get_list = 1;
    }
    break;
case 6:
    if (state == 0) {
        printf("*-----system-----*\n");
        printf("Please connect first\n");
    }
    else
    {
        if (get_list == 0)
        {
            printf("*-----system-----*\n");
            printf("Please get client list first\n");
        }
        else
        {
            SendMessage();
            lock_print(4);
        }
    }
    break;
case 7:
    if (state == 1)
        Close();
    WSACleanup();

```

```

    return 0;        //不用break;
default:
    break;
}
//处理完客户的选择，然后把多的指示消息一口气打印出来，如果有的话
pthread_mutex_lock(&Queue_mutex);
while (Client_msgQueue.size() != 0)
{
    Client_msgQueue.front().print_msg();
    Client_msgQueue.pop();
}
pthread_mutex_unlock(&Queue_mutex);
choose = Screen(false);
}

```

首先是不断的根据用户给的七个选项进行不同的处理

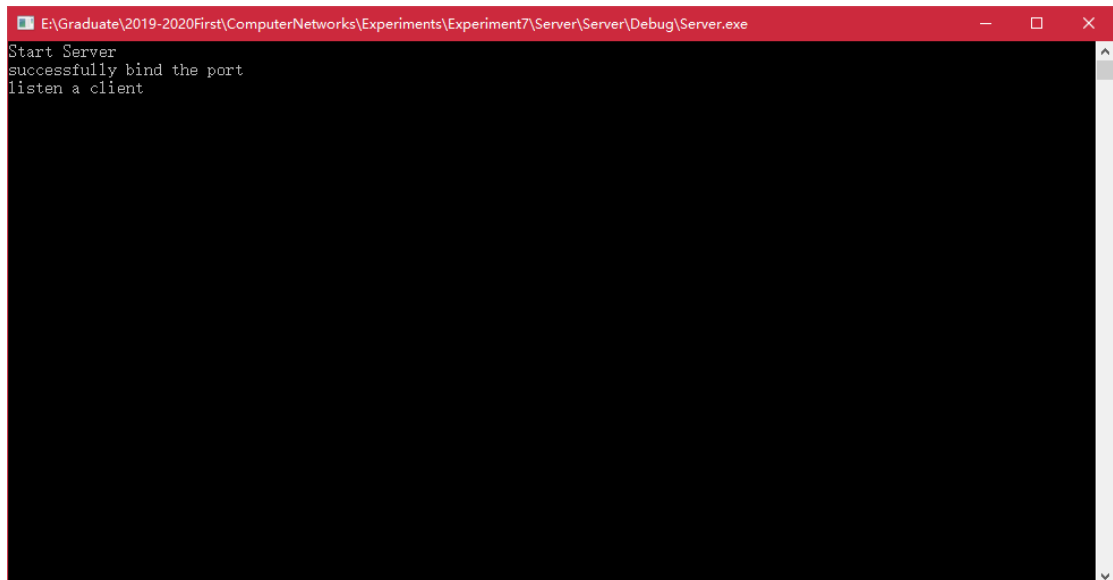
另外，在循环的最后，需要每次循环都去查询消息队列看是否存在新的消息进来

- 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）

```
while (true) {
    pthread_testcancel();
    memset(BUFFER, 0, LEN);
    memset(tmp, 0, LEN);
    if ((Length = recv(sListen, tmp, LEN, 0)) > 0) {
        int i;
        int type = gettype(tmp[0]);
        for (i = 1; i < Length &&!(tmp[i] == '$' &&tmp[i + 1] == '@'); ++i)
            BUFFER[i-1] = tmp[i];
        BUFFER[i] = '\0';
        struct msg m = msg(type, BUFFER, i);
        //printf("REC : \n%s\n", BUFFER);
        pthread_mutex_lock(&Queue_mutex);{
            Client_msgQueue.push(m);          //增加一条消息
        }
        pthread_mutex_unlock(&Queue_mutex);
    }
    else //比如说服务器端单方面的关闭了连接，会造成这种情况，当然网络在物理层面断了也是会发生的
    {
        printf("*-----system-----*\n");
        printf("the connect has been canceled\n");
        state = 0;
        pthread_exit(NULL);
    }
}
```

不断调用 recv 函数监听，如果收到一个数据包，则进行处理，并取得锁，然后 push 进消息队列

- 服务器初始运行后显示的界面



服务器端反正没有什么界面美化要求，丑一点就丑一点吧

- 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）

```

while ((sServer = accept(sListen, (sockaddr*)&saClient, &len)) != INVALID_SOCKET) {
    // ...

    // ...
    printf("A new client is connected, the id is %d\n", ClientList.size());
    struct Client CL = Client();
    CL.ID = ClientList.size();
    CL.socket = sServer;
    CL.state = 1;
    strcpy(CL.IP, inet_ntoa(saClient.sin_addr));
    ClientList.push_back(CL);
    pthread_create(&CL.ClientThread, NULL, Client_thread, (void*)CL.ID);
}

```

不断的调用 `accept`，当有新的客户端接入，就增加一个新的 `client` 实例到队列中，并创建对应的子线程

- 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）

```

while (true) {
    memset(BUFFER, 0, LEN);
    memset(tmp, 0, LEN);
    if ((Length = recv(cl.socket, tmp, LEN, 0)) > 0) {
        printf("receive a message from client %d\n", id);
        int i;
        for (i = 0; i < Length - 1 && !(tmp[i] == '$' && tmp[i + 1] == '@'); ++i)
            BUFFER[i] = tmp[i];
        BUFFER[i] = '\0';
        if (BUFFER[0] == 'C') {
            printf("Get a close requirement from client %d\n", id);
            ClientList[id].state = 0;
            pthread_exit(NULL);
        }
        else {
            Response(BUFFER, id);
        }
    }
    else {
        printf("Close connect with client %d\n", id);
        ClientList[id].state = 0;
        pthread_exit(NULL);
    }
}

```

大概就跟客户端的差不多

但是需要加入一个判断是否是 `close` 请求报文，如果是，则直接关闭这个实例并关闭这个子线程

如果不是则调用 `response` 函数分别对不同的请求报文进行相应

- 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:

```
E:\Graduate\2019-2020First\ComputerNetworks\Experiments\Experiment7\Client\Client\Debug\Client.exe
#
#      1:connect
#      2:close
#      3:get time info
#      4:get name info
#      5:get client list
#      6:sent massage
#      7:exit
#
#####
*-----system-----*
please enter your choose:
1
*-----system-----*
Your choice is: 1 : connect
*-----system-----*
Please enter the target IP:
10.15.62.128
*-----system-----*
Please enter the target Port:
0441
*-----system-----*
Start connect
*-----system-----*
connect success
*-----system-----*
sub_thread create success!
*-----system-----*
please enter your choose:
```

服务器端:

```
C:\Users\WangYan\Desktop\lab7\Server.exe
Start Server
successfully bind the port
listen a client
A new client is connected, the id is 0
-
```

Wireshark 抓取的数据包截图:

客户端:

1	0.000000	10.180.125.197	10.15.62.128	TCP	66 63887 → 441 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001995	10.15.62.128	10.180.125.197	TCP	66 441 → 63887 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.002103	10.180.125.197	10.15.62.128	TCP	54 63887 → 441 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4	0.004128	10.15.62.128	10.180.125.197	TCP	62 441 → 63887 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=8
5	0.044216	10.180.125.197	10.15.62.128	TCP	54 63887 → 441 [ACK] Seq=1 Ack=9 Win=131328 Len=0

服务器:

1	0.000000	10.180.125.197	10.15.62.128	TCP	66 63972 → 441 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000094	10.15.62.128	10.180.125.197	TCP	66 441 → 63972 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.001644	10.180.125.197	10.15.62.128	TCP	60 63972 → 441 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4	0.003692	10.15.62.128	10.180.125.197	TCP	62 441 → 63972 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=8
5	0.046447	10.180.125.197	10.15.62.128	TCP	60 63972 → 441 [ACK] Seq=1 Ack=9 Win=131328 Len=0

- 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：（emmm 这边还能够看到上面服务器端发送的 hello）

```
##### received message #####
Hello
*-----system-----*
please enter your choose:
3
*-----system-----*
Your choice is: 3 : get time info
##### received message #####
Your ID:1
2019-10-16 16:38:12
*-----system-----*
please enter your choose:
```

服务器端：

```
C:\Users\WangYan\Desktop\lab7\Server.exe
Start Server
successfully bind the port
listen a client
A new client is connected, the id is 0
A new client is connected, the id is 1
receive a message from client 1
Client 1 is asking for current time
send time successfully
```

会显示一个某个客户端请求时间消息和发送时间消息成功的消息

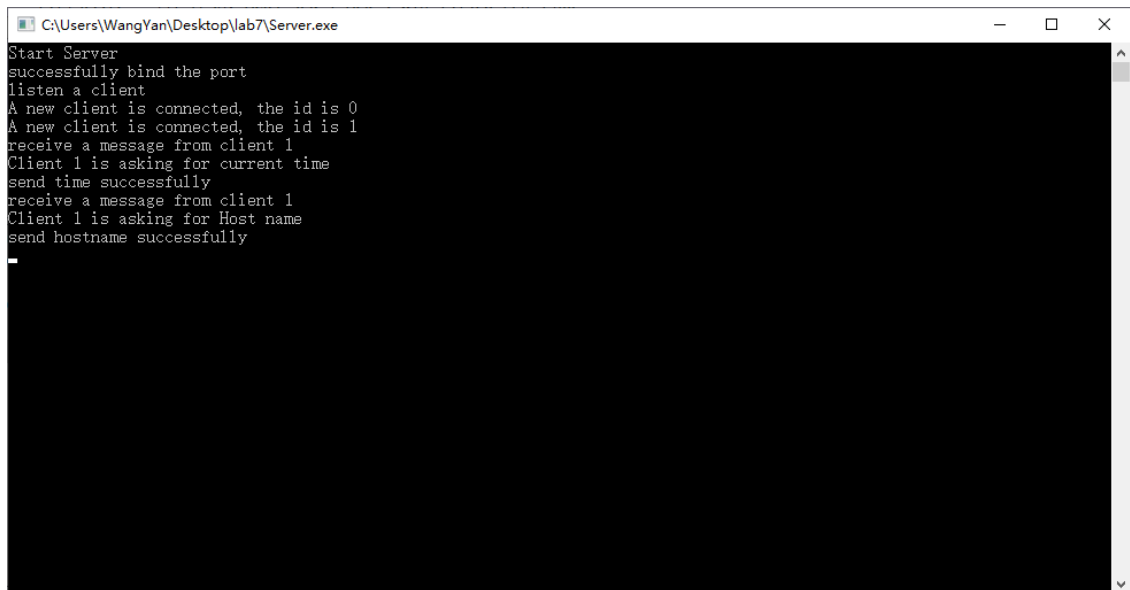
（考虑到需要传送数据给另一个客户端，所以我这边开了两个）

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


```
please enter your choose:
4
*-----system-----*
Your choice is: 4 : get name info
##### received message #####
Your ID:1
DESKTOP-L05KD06
*-----system-----*
please enter your choose:
```

服务器端



```
C:\Users\WangYan\Desktop\lab7\Server.exe
Start Server
successfully bind the port
listen a client
A new client is connected, the id is 0
A new client is connected, the id is 1
receive a message from client 1
Client 1 is asking for current time
send time successfully
receive a message from client 1
Client 1 is asking for Host name
send hostname 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）：

> Frame 14: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0 > Ethernet II, Src: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c), Dst: JuniperN_67:28:7c (88:e0:f3:67:28:7c) > Internet Protocol Version 4, Src: 10.180.125.197, Dst: 10.15.62.128 > Transmission Control Protocol, Src Port: 63972, Dst Port: 441, Seq: 1461, Ack: 41, Len: 1460 > Data (1460 bytes)		
Data: 4e244000... [Length: 1460]		
0030	02 01 1b c2 00 00 4e 24 40 00 00 00 00 00 00 00N\$ @.....
0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 Length (data len)		

上图是客户端的报文

就一个申请字段 N 和一个结束字段\$@

> Frame 15: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0 > Ethernet II, Src: JuniperN_67:28:7c (88:e0:f3:67:28:7c), Dst: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c) > Internet Protocol Version 4, Src: 10.15.62.128, Dst: 10.180.125.197 > Transmission Control Protocol, Src Port: 441, Dst Port: 63972, Seq: 41, Ack: 2921, Len: 28 > Data (28 bytes)		
Data: 4e596f75722049443a310a4445534b544f502d4c30354b44... [Length: 28]		
0000	48 45 20 a6 d7 7c 88 e0 f3 67 28 7c 08 00 45 00	HE ·g(..E·
0010	00 44 78 00 40 00 3d 06 f4 ab 0a 0f 3e 80 0a b4	·Dx·@·=· ····>···
0020	7d c5 01 b9 f9 e4 4a 49 2c 6c c4 13 fc 75 50 18	}·····JI ,1···uP·
0030	04 02 0c ed 00 00 4e 59 6f 75 72 20 49 44 3a 31	·····NY our ID:1
0040	0a 44 45 53 4b 54 4f 50 2d 4c 30 35 4b 44 30 36	·DESKTOP -L05KD06
0050	24 40	\$@

上图是响应报文，包括

类型字段 N

Id 字段 “Your ID: 1”

以及数据字段，服务器主机名称

和结束字段\$@

相关的服务器的处理代码片段：

```
case 2:
    printf("Client %d is asking for Host name\n", ID);
    strcat(SendBuffer, "N");
    strcat(SendBuffer, "Your ID:");
    strcat(SendBuffer, sourceid);
    strcat(SendBuffer, "\n");
    gethostname(tmpBuffer, 128);
    strcat(SendBuffer, tmpBuffer);
    strcat(SendBuffer, "$@");
    if (send(cl.socket, SendBuffer, strlen(SendBuffer), 0) == SOCKET_ERROR) {
        printf("send hostname error with error code: %d\n", WSAGetLastError());
    }
    else
        printf("send hostname successfully\n");
    break;
```

用 gethostname 获取主机名称

其他就是组报文和发送报文，应该内容很清楚

- 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：

```
please enter your choose:
5
*-----system-----*
Your choice is: 5 : get client list
##### received message #####
Your ID:1
ID:0&&IP:10.180.125.197&&Port63887
ID:1&&IP:10.180.125.197&&Port63972
*-----system-----*
please enter your choose:
```

服务器端：


```
receive a message from client 1
Client 1 is asking for Client list
send list 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）：

请求报文：

The screenshot shows a Wireshark packet capture of a request packet (Frame 17). The packet details pane on the left shows the following structure:

- > Frame 17: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
- > Ethernet II, Src: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c), Dst: JuniperN_67:28:7c (88:e0:f3:67:28:7c)
- > Internet Protocol Version 4, Src: 10.180.125.197, Dst: 10.15.62.128
- > Transmission Control Protocol, Src Port: 63972, Dst Port: 441, Seq: 2921, Ack: 69, Len: 1460
- ▼ Data (1460 bytes)

The data field is expanded, showing a hex dump of the payload. The first few bytes are 4c 24 40 00 00 00 00 00 00 00 00 00 00 00 00 00, which correspond to the ASCII string "L\$ @". The rest of the data is represented by a series of dots in the hex dump, indicating a large amount of data (1460 bytes total).

Offset	Hex	ASCII
0030	02 01 17 f2 00 00 4c 24L\$@.....
0040	40 00 00 00 00 00 00 00
0050	00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00
0070	00 00 00 00 00 00 00 00
0080	00 00 00 00 00 00 00 00
0090	00 00 00 00 00 00 00 00
00a0	00 00 00 00 00 00 00 00
00b0	00 00 00 00 00 00 00 00
00c0	00 00 00 00 00 00 00 00
00d0	00 00 00 00 00 00 00 00

响应报文：

```

Frame 18: 137 bytes on wire (1096 bits), 137 bytes captured (1096 bits) on interface 0
Ethernet II, Src: JuniperN_67:28:7c (88:e0:f3:67:28:7c), Dst: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c)
Internet Protocol Version 4, Src: 10.15.62.128, Dst: 10.180.125.197
Transmission Control Protocol, Src Port: 441, Dst Port: 63972, Seq: 69, Ack: 4381, Len: 83
Data (83 bytes)
  Data: 4c596f75722049443a310a49443a30262649503a31302e31...
  [Length: 83]

```

```

0000 48 45 20 a6 d7 7c 88 e0 f3 67 28 7c 08 00 45 00 HE ..|.. .g(|...E.
0010 00 7b 78 01 40 00 3d 06 f4 73 0a 0f 3e 80 0a b4 .{x.@=-. .s...>...
0020 7d c5 01 b9 f9 e4 4a 49 2c 88 c4 14 02 29 50 18 }.....JI ,....)P.
0030 04 02 c1 56 00 00 4c 59 6f 75 72 20 49 44 3a 31 ...V...LY our ID:1
0040 0a 49 44 3a 30 26 26 49 50 3a 31 30 2e 31 38 30 .ID:0&&I P:10.180
0050 2e 31 32 35 2e 31 39 37 26 26 50 6f 72 74 36 33 .125.197 &&Port63
0060 38 38 37 0a 49 44 3a 31 26 26 49 50 3a 31 30 2e 887.ID:1 &&IP:10.
0070 31 38 30 2e 31 32 35 2e 31 39 37 26 26 50 6f 72 180.125. 197&&Por
0080 74 36 33 39 37 32 0a 24 40 t63972.$ @

```

蓝色部分即响应报文的内容

其中开头的 L 表示类型是返回 list

之后是请求客户端的 id

随后按照 ID、IP、port 的顺序依次传送每个客户端的信息

最后\$@表示结束

相关的服务器的处理代码片段：

```

case 3:
    printf("Client %d is asking for Client list\n", ID);
    strcat(SendBuffer, "L");
    strcat(SendBuffer, "Your ID:");
    strcat(SendBuffer, sourceid);
    strcat(SendBuffer, "\n");
    for (int i = 0; i < ClientList.size(); ++i)
    {
        memset(id, 0, 5);
        memset(port, 0, 128);
        sockaddr_in tmpaddr;
        int tmpen = sizeof(sockaddr_in);
        getpeername(ClientList[i].socket, (sockaddr*)&tmpaddr, &tmpen);
        _itoa(i, id, 10);
        int P = ntohs( tmpaddr.sin_port );
        _itoa(P, port, 10);
        strcat(SendBuffer, "ID:");
        strcat(SendBuffer, id);
        strcat(SendBuffer, "&&IP:");
        strcat(SendBuffer, ClientList[i].IP);
        strcat(SendBuffer, "&&Port");
        strcat(SendBuffer, port);
        strcat(SendBuffer, "\n");
    }
    strcat(SendBuffer, "$@");
    //printf("List info is %s\n", SendBuffer);
    if (send(cl.socket, SendBuffer, strlen(SendBuffer), 0) == SOCKET_ERROR) {
        printf("send list error with error code: %d\n", WSAGetLastError());
    }
    else
        printf("send list successfully\n");
    break;

```

具体含义是

首先给出报文头部

其次 for 循环内查找所有的客户端列表，并将他们的 id、ip、port 等信息依次组装起来

最后加上结束标志并发送

- 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：

```

*-----system-----*
please enter your choose:
6
*-----system-----*
Your choice is: 6 : sent message
*-----system-----*
please choose the Client number:
0
*-----system-----*
please choose the message to send(no more than 1200):
123456
##### received message #####
Your ID:1
Success
*-----system-----*
please enter your choose:

```

会收到一个成功的返回消息

服务器:

```
receive a message from client 1
Client 1 is asking for send message
The destination Client is 0
send message to destination successfully
send message to source successfully
```

上图显示客户端 1 想要发送消息给客户端 0

然后服务器成功向目的客户端和源客户端发送消息（给目的客户端是源客户端发送的消息，给源客户端发送一条成功的消息）

接收消息的客户端:

```
*-----system-----*
Your choice is: 3 : get time info
##### received message #####
Hello
##### received message #####
Source ID:1
123456
##### received message #####
Your ID:0
2019-10-16 18:22:10
*-----system-----*
please enter your choose:
```

上图显示了客户端 0 收到服务器发送的 hello 消息

以及从客户端 0 发送的 123456 消息

注:

因为客户端主线程需要读取用户的选择，而实验要求里面写到:

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

也就是说，主线程必然阻塞，所以要显示另一个客户端发送的消息，需要主线程发送别的消息之后再进行读取

Wireshark 抓取的数据包截图（发送和接收分别标记）:

```

> Frame 46: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
> Ethernet II, Src: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c), Dst: JuniperN_67:28:7c (88:e0:f3:67:28:7c)
> Internet Protocol Version 4, Src: 10.180.125.197, Dst: 10.15.62.128
> Transmission Control Protocol, Src Port: 64421, Dst Port: 441, Seq: 4381, Ack: 152, Len: 1460
▼ Data (1460 bytes)
  Data: 4d3026263132333435362440000000000000000000000000...
  [Length: 1460]

```

0030	02 00 9c f6 00 00 4d 30 26 26 31 32 33 34 35 36M0 &&123456
0040	24 40 00 00 00 00 00 00 00 00 00 00 00 00 00	\$@-.....
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

上图是发送消息

就是一个开头，一个目的端口 0，以及内容 123456（其中\$\$用于分割），以及结束标识\$@

接受消息：

发送端接收的消息

```

> Frame 114: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: JuniperN_67:28:7c (88:e0:f3:67:28:7c), Dst: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c)
> Internet Protocol Version 4, Src: 10.15.62.128, Dst: 10.180.125.197
> Transmission Control Protocol, Src Port: 441, Dst Port: 64532, Seq: 152, Ack: 5841, Len: 20
▼ Data (20 bytes)
  Data: 4d596f75722049443a310a537563636573732440
  [Length: 20]

```

0000	48 45 20 a6 d7 7c 88 e0 f3 67 28 7c 08 00 45 00	HEg(..E-
0010	00 3c 78 27 40 00 3d 06 f4 8c 0a 0f 3e 80 0a b4	.<x'@.=.>...
0020	7d c5 01 b9 fc 14 ff b3 72 7e 5c 03 f7 af 50 18	}..... r~\...P.
0030	04 02 e9 c6 00 00 4d 59 6f 75 72 20 49 44 3a 31MY our ID:1
0040	0a 53 75 63 63 65 73 73 24 40	.Success \$@

发送端接受到一个显示发送成功的报文，开头是类型字段，随后是 ID 字段，最后一个 success 和结尾的消息

接收端收到的消息：

```
> Frame 113: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
> Ethernet II, Src: JuniperN_67:28:7c (88:e0:f3:67:28:7c), Dst: IntelCor_a6:d7:7c (48:45:20:a6:d7:7c)
> Internet Protocol Version 4, Src: 10.15.62.128, Dst: 10.180.125.197
> Transmission Control Protocol, Src Port: 441, Dst Port: 64530, Seq: 9, Ack: 1, Len: 21
▼ Data (21 bytes)
  Data: 4d536f757263652049443a310a3132333435362440
      [Length: 21]
```

0000	48 45 20 a6 d7 7c 88 e0 f3 67 28 7c 08 00 45 00	HEg(..E.
0010	00 3d 78 26 40 00 3d 06 f4 8c 0a 0f 3e 80 0a b4	..=x&@..=..>...
0020	7d c5 01 b9 fc 12 83 81 01 99 10 cc bb 71 50 18	}.....qP.
0030	04 02 8d 09 00 00 4d 53 6f 75 72 63 65 20 49 44MS ource ID
0040	3a 31 0a 31 32 33 34 35 36 24 40	:1.12345 6\$@

消息内容以 M 类型开头，随后是 sourceid 接下来是内容 123456 和结束符\$@

相关的服务器的处理代码片段：

先是对目的客户端发送消息

其中加入了三个错误的处理

一个是发送的目的端口超出了范围，一个是目的端口已经关闭，还有一个是自己向自己发送消息

```

case 4:
    strcat(SendBuffer, "M");
    strcat(SendBuffer, "Source ID:");
    strcat(SendBuffer, sourceid);
    strcat(SendBuffer, "\n");
    for (i = 1; !(BUFFER[i] == '&' && BUFFER[i + 1] == '&'); ++i) {
        desid[i - 1] = BUFFER[i];
    }
    //strcat(SendBuffer, desid);
    strcat(SendBuffer, BUFFER+i+2);
    strcat(SendBuffer, "$@");
    destination = atoi(desid);
    printf("Client %d is asking for send message\nThe destination Client is %d\n", ID, destination);
    if (destination > ClientList.size() || destination < 0) {
        printf("The destination is out of range\n");
        sendflag = false;
        error_code = 2;
    }
    else {
        cld = ClientList[destination];
        //向目的client发送消息
        if (cld.state == 0) {
            printf("The destination has closed\n");
            sendflag = false;
            error_code = 3;
        }
        else if (destination == ID) {
            printf("The source id equals to the destination id\n");
            sendflag = false;
            error_code = 4;
        }
        else
        {
            if (send(cld.socket, SendBuffer, strlen(SendBuffer), 0) == SOCKET_ERROR) {
                printf("send message to destination error with error code: %d\n", WSAGetLastError());
                sendflag = false;
                error_code = WSAGetLastError();
            }
            else
                printf("send message to destination successfully\n");
        }
    }
}

```

随后是向来源客户端发送消息

需要根据当前的错误码向来源站发送消息或者发送成功的消息

```

//向来源client发送消息
memset(SendBuffer, 0, LEN);
strcat(SendBuffer, "M");
strcat(SendBuffer, "Your ID:");
strcat(SendBuffer, sourceid);
strcat(SendBuffer, "\n");
if (sendflag == true)
    strcat(SendBuffer, "Success$@");
else {
    strcat(SendBuffer, "Failed with error code:");
    _itoa(error_code, Error_code, 10);
    strcat(SendBuffer, Error_code);
    if (error_code == 2) {
        strcat(SendBuffer, " The destination is out of range");
    }
    else if (error_code == 3) {
        strcat(SendBuffer, " The destination has closed");
    }
    else if (error_code == 4) {
        strcat(SendBuffer, " The source id equals to the destination id");
    }
    strcat(SendBuffer, "$@");
}
if (send(cl_socket, SendBuffer, strlen(SendBuffer), 0) == SOCKET_ERROR) {
    printf("send message to source error with error code: %d\n", WSAGetLastError());
}
else
    printf("send message to source successfully\n");
break;

```

相关的客户端（发送和接收消息）处理代码片段：

发送消息：

要求用户输入目的的客户端口和消息内容

```

void SendMessage() {
    char ask[LEN];
    char message[1200];
    char idchar[5];
    int id;
    memset(ask, 0, LEN);
    ask[0] = 'M';
    printf("*-----system-----*\n");
    printf("please choose the Client number:\n");
    scanf("%d", &id);
    _itoa(id, idchar, 10);
    printf("*-----system-----*\n");
    printf("please choose the message to send(no more than 1200):\n");
    scanf("%s", &message);
    strcat(ask, idchar);
    strcat(ask, "&&");
    //因为需要进行目的的client和报文message的分割
    //但是因为strcat之后，得到的char应该是长度不固定的，所以需要进行分割
    strcat(ask, message);
    strcat(ask, "$@");
    send(sListen, ask, LEN, 0);
}

```

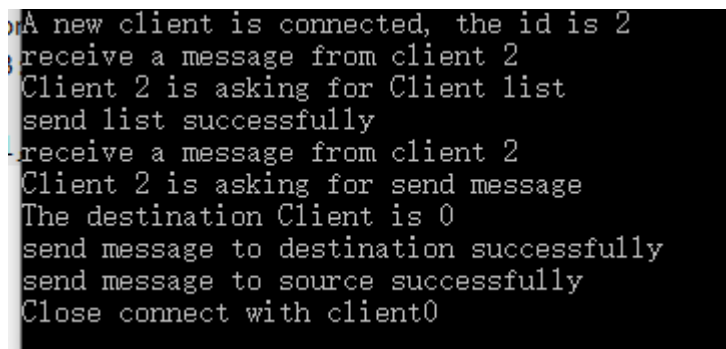

接受消息的处理跟其他的消息都一样，就是打印，我就不发了：)

- 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。

客户端直接弹出了连接中断的消息，但服务器端没有反应，没有发出 TCP 连接释放的消息（可能跟我直接右上角点叉有关系）

服务器端好像没有发生变化

- 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？



```
A new client is connected, the id is 2
receive a message from client 2
Client 2 is asking for Client list
send list successfully
receive a message from client 2
Client 2 is asking for send message
The destination Client is 0
send message to destination successfully
send message to source successfully
Close connect with client0
```

一个新的连上了，然后向客户端 0 发送消息，然后成功发送了消息，但是发完发现客户端 0 掉线了，于是显示已经关闭了跟客户端 0 的连接

- 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。

服务器正常收到了 100 个响应，并进行了处理。见下图（但是太长了，我没全都截图）

客户端:

```
send time successfully
receive a message from client 3
Client 3 is asking for current time
send time successfully
receive a message from client 3
Client 3 is asking for current time
send time successfully
receive a message from client 3
Client 3 is asking for current time
send time successfully
receive a message from client 3
Client 3 is asking for current time
send time successfully
receive a message from client 4
Client 4 is asking for current time
send time successfully
receive a message from client 4
Client 4 is asking for current time
send time successfully
receive a message from client 4
Client 4 is asking for current time
send time successfully
receive a message from client 4
Client 4 is asking for current time
send time successfully
receive a message from client 4
Client 4 is asking for current time
```

六、 实验结果与分析

根据你编写的程序运行效果，分别解答以下问题（看完请删除本句）：

- 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？
不需要调用 bind
源端口是系统自动分配的
客户端的端口发生了变化
- 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？
可以
- 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？
不完全一致
- 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？

根据 socket 套接字的信息，不同客户端的端口实际上是不一样的

- 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）

原来的是 `established` 后来消失了

- 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？

没有变化

换句话说如果突然掉线还不知道断掉了

应该定时向客户端发送一个简短的消息来检测连接是否还在

（PS：感觉可以考虑的方式是，检测之后增加一个 `buffer`，每次收到一个目的客户端已经断掉连接的就缓存

但是这样因为每次端口不同，每次 `ip` 不同，还需要增加 `cookie` 去作为客户端识别）

七、 讨论、心得

我遇到的一个最最坑爹的错误来自于学号问题

因为端口号要学号开头，而我的学号后四位是 0 开头的

于是按照正常人的思维，直接服务器端就定义一个宏然后令它等于 `0441`

但是问题在于 c 语言中 **八进制开头也是以 0 开头的**，于是在客户端输入目标服务器端口试图连接这一步，死活连不上（因为设置了输入格式为 `%d`），但是表示测试半天，还以为是别的地方出了问题

讲真，连得上，调试什么的，无非就是碰到啥格式问题，`printf` 一下下就好了，但是，连不上就很懵逼了

教训就是这个了，建议就是，请务必把这个坑写进实验提示里面，不然后面的人一定也会踩这个坑

第二个问题在于 `pthread` 的 `cancel` 竟然只是发送一个要子线程关闭的消息，需要子线程自己设置如

何关闭线程这件事情——当然这事情，发现了其实还好，网上查查解决方案就有了

第三个教训来自于符号优先级

就是 `while` 循环里面的循环条件，包括 `for` 循环，总是多个判断符号，需要注意其中的逻辑和符号的优先级，因为这个问题，也造成了 `client` 显示连上了，但是 `server` 表示没连上的一个奇葩现象（起因是，需要把前面一个判断条件加上一个括号，增加优先级，否则得到的客户端的套接字的值其实是后面条件判断的 `bool` 值），另外报的错则是判断报文结束的地方。

至于其他的，总之一句话，经常性的按照写 C++ 的逻辑去处理一些事情（其实是 C with Class and STL 吧），要记住 C 是更加偏向底层的语言

啊对，还有就是 `getsockname` 和 `getpeername` 不要弄反了

最后的最后，我必须必须吐槽一句，`pthread` 的 `dll` 的设计真的是非常的让人崩溃

为什么，`x86` 文件夹下面的要放到 `sysWOW64`

而 `x64` 下的 `dll` 文件需要放到 `system32` 下？

是道德的沦丧还是人性的泯灭？？害的我查了半天的错误