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：	姚熙源
学　　院：	计算机学院
系：	计算机科学与技术
专　　业：	软件工程
学　　号：	3190300677
指导教师：	董玮

2022 年 3 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：-　实验地点：计算机网络实验室

一、 实验目的

- 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
- **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
- Visual C++、gcc 等 C++集成开发环境。

四、 操作方法与实验步骤

- 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()，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：调用 send()将获取时间请求发送给服务器，**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：调用 send()将获取名字请求发送给服务器，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：调用 send()将获取客户端列表信息请求发送给服务器，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，然后调用 send()将数据发送给服务器，观察另外一个客户端是否收到数据。
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，并调用 send()发给客户端
2. 请求类型为获取名字：调用 GetComputerName 获取本机名，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、端口等数据通过调用 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，将要转发的消息组

装通过调用 send()发给接收客户端（使用接收客户端的 socket 句柄）。

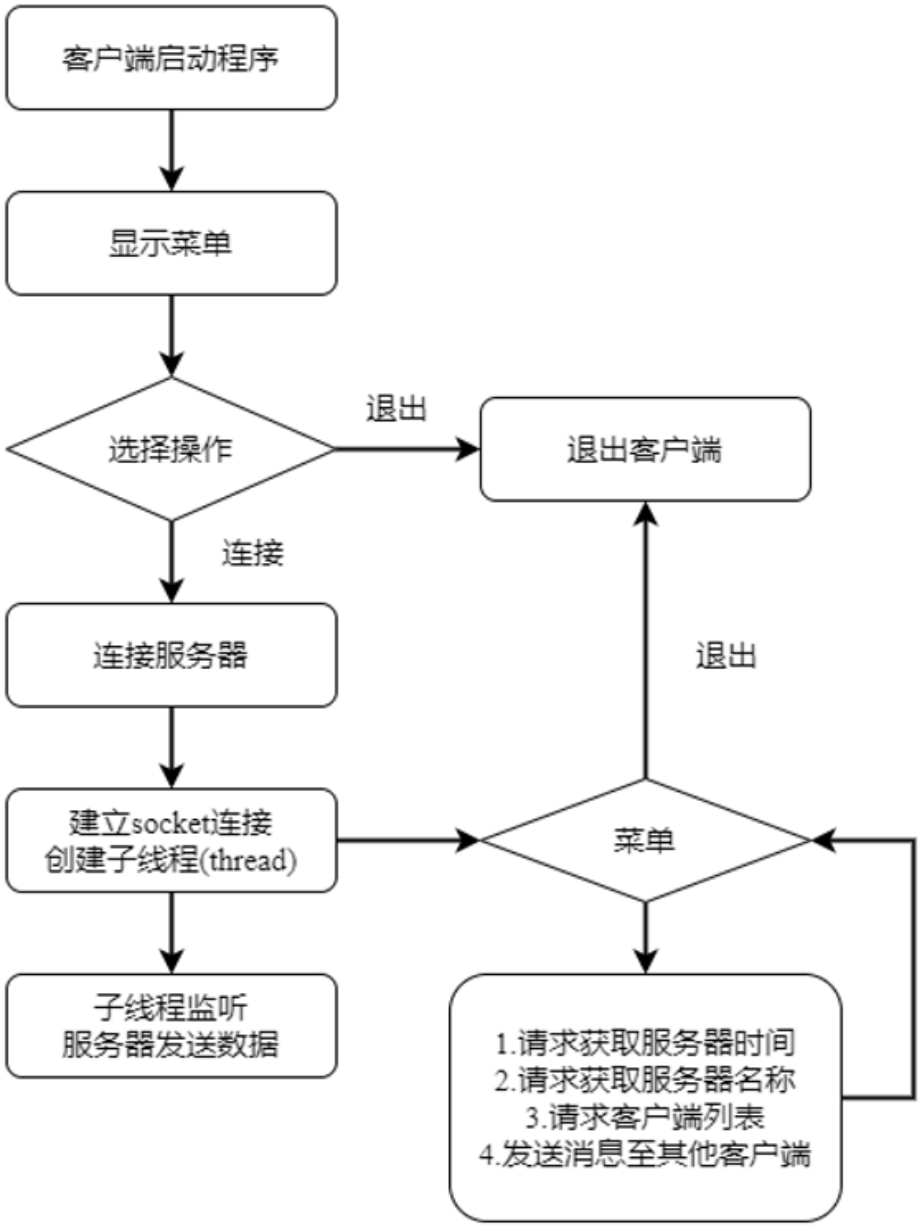
- 编程结束后，双方程序运行，检查是否实现功能要求，如果有问题，查找原因，并修改，直至满足功能要求
- 使用多个客户端同时连接服务端，检查并发性

五、 实验数据记录和处理

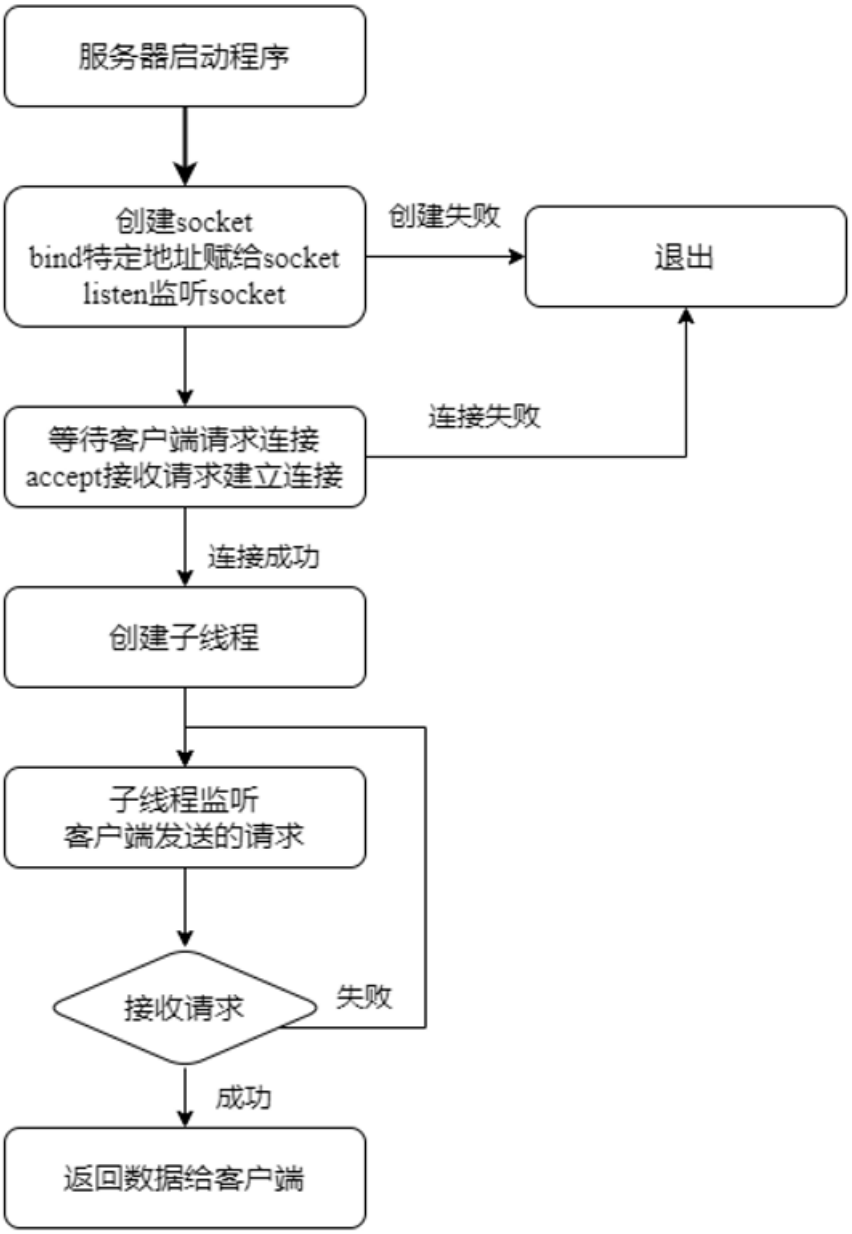
请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **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**
- 客户端和服务器端框架图（用流程图表示）

客户端框架图



服务器框架图



开启服务端和客户端说明：

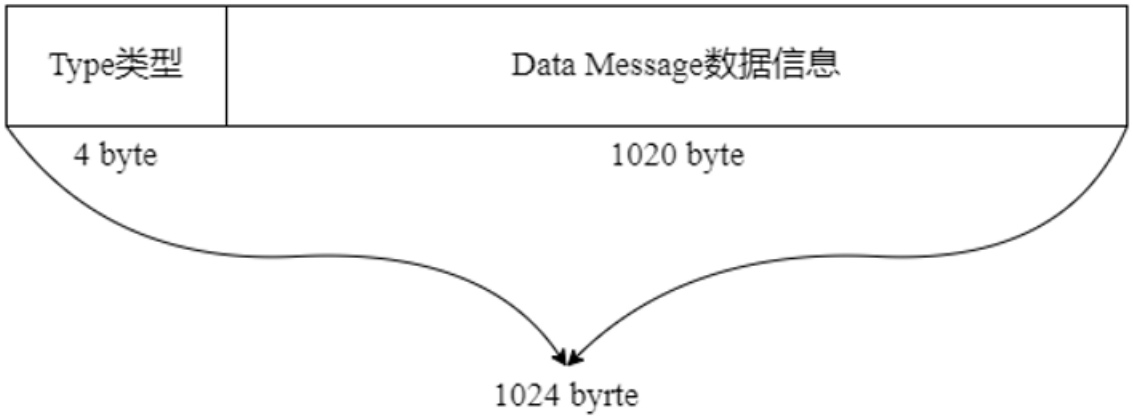
Linux 操作系统命令行编译 C++程序

g++ -o server server.cpp -lpthread

g++ -o client client.cpp -lpthread

Linux 操作系统命令行启动程序 ./client 或 ./server

- 客户端请求数据包格式和类型定义



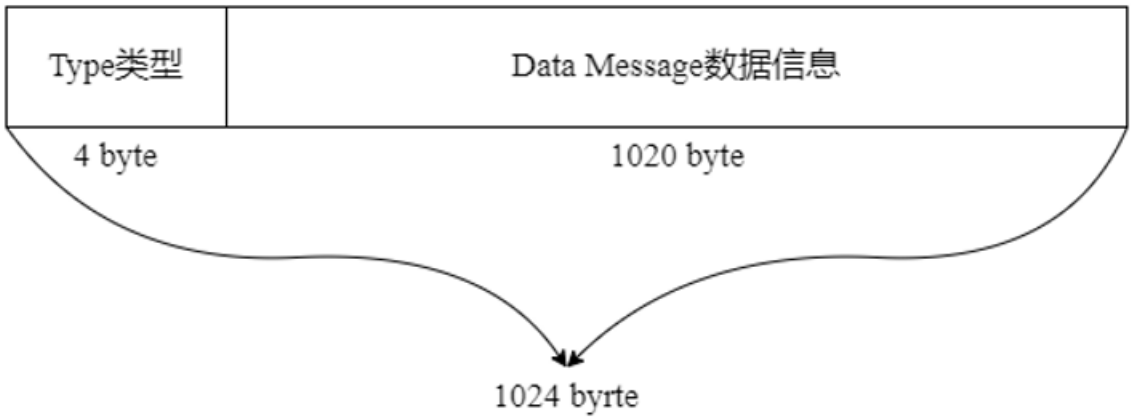
time 类型：客户端向服务端发送请求时间数据包，“time”

name 类型：客户端向服务端发送请求服务端名称数据包，“name”

list 类型：客户端向服务端发送请求客户端列表信息数据包，“list”

send 类型：客户端向服务端发送消息数据包（“send#目标客户端 ID#消息”），由服务端再向目标客户端发送消息，以’#’字符来区分目标客户端 ID 和发送的消息。

- 服务端返回数据包格式和类型定义



TIME 类型：服务端返回时间信息数据包，“TIMExx:xx:xx”

NAME 类型：服务端返回名称信息数据包，“NAMExxxxxx…”

LIST 类型：服务端返回客户端列表信息数据包，“LISTxxxxxxx…”

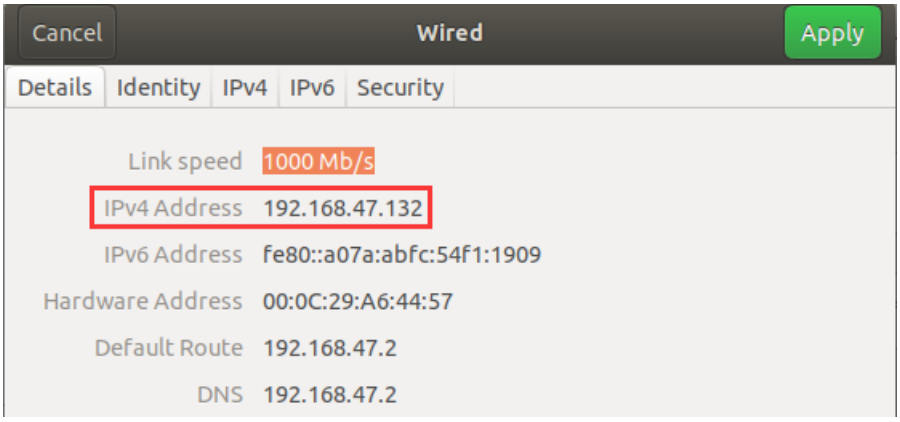
SEND 类型：服务端将客户端的消息发送到目标客户端，Data Message 信息：“SENDID: %d, Address: %s, Port: %d Sent A Message To You: %s”。

- 客户端初始运行后显示的菜单选项

  客户端初运行后显示半完整菜单，连接服务器才显示全部的菜单列表。连接服务器 IP 地址是 192.168.47.132（图中红色框）

```
peter@peter-virtual-machine:~/cn$ ./client
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|                Menu               |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 7. Exit Client                    |
+-----------------------------------+
1
Enter Server IP :
192.168.47.132
Enter Server Port :
3677
Connect Server Success!
[Packet Received] The Packet Is Connection Packet
[Received Message] : Reply Message From Server!

Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|                Menu               |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 3. Request Server Time            |
| 4. Request Server Name            |
| 5. Request Client List            |
| 6. Send Message To Other Client   |
| 7. Exit Client                    |
+-----------------------------------+
```



- 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){
        packetType = -1;
        memset(serverResponse, 0, BUFFER_LEN);
        int res = recv(sockfd, &serverResponse, sizeof(serverResponse), 0);
        if(res > 0){
            string packet = "Connection Packet";
            if(!strncmp(serverResponse, "TIME", 4)){
                packetType = 0;//time packet type
                packet = "Time Packet";
            }
            else if(!strncmp(serverResponse, "NAME", 4)){
                packetType = 1;
                packet = "Name Packet";
            }
            else if(!strncmp(serverResponse, "LIST", 4)){
                packetType = 2;
                packet = "Client List Packet";
            }
            else if(!strncmp(serverResponse, "SEND", 4)){
                packetType = 3;
                packet = "Send Message Packet";
            }
            cout << "[Packet Received] The Packet Is " << packet << endl;
            cout << "[Received Message] : " << ( (packetType==-1) ?
serverResponse : (serverResponse+4) )<< endl;
        }
        else if(res < 0){ //receive fail
            pthread_exit(NULL);
        }
        else {// == 0
            cout << "Error : Socket Closed! Exiting Current Thread!" << endl;
            pthread_exit(NULL);
        }
}
```

利用 recv 函数来接收从服务器发来的数据包信息，接收到信息则返回大于 0，接收不到就返回 0 或是负数（一般是-1），指连接断开了然后退出线程。接收到后就判断是哪种类型的数据包，然后在客户端输出信息。

- 服务器初始运行后显示的界面
  - 初始运行后

```
peter@peter-virtual-machine:~/cn$ ./server
[Socket Created] Sucess!
[Bind Socket] Success!
[Listen Socket] Success!
```

  - 客户端连接服务器后

```
peter@peter-virtual-machine:~/cn$ ./server
[Socket Created] Sucess!
[Bind Socket] Success!
[Listen Socket] Success!
[Connection Established] Client Address : 192.168.47.132 PORT : 46738
Reply Message Sent Successfully!
```

  - 客户端发送请求后，服务端响应请求

```
peter@peter-virtual-machine:~/cn$ ./server
[Socket Created] Sucess!
[Bind Socket] Success!
[Listen Socket] Success!
[Connection Established] Client Address : 192.168.47.132 PORT : 46738
Reply Message Sent Successfully!
Data Packet Receive From Client : time
Time Data Has Been Sent To CLient 1!
Data Packet Receive From Client : name
Server Name Has Been Sent To CLient 1!
```

- 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）

  客户端在连接服务端时使用 pthread_create()函数创建子线程，每次请求服务端都会创建一个新的线程进行操作（比如请求服务端时间、名称等信息）。

```
while(true){
        //accept sock
        clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress,
&clientAddressSize);
        if(clientSocket == -1){
            //accept fail
            perror("Error : Accept Failed!\n");
            continue;
        }
        else{
            //accept success
            cout << "Connection Established! Client Address : " <<
inet_ntoa(clientAddress.sin_addr)
                 << " PORT : " << ntohs(clientAddress.sin_port) << endl;
             addrList[ID] = (char*)malloc(sizeof(char) * ADDRESS_LEN); //
request size
```

```
            strcpy(addrList[ID], inet_ntoa(clientAddress.sin_addr));
            portList[ID] = ntohs(clientAddress.sin_port);
            isConnect[ID] = 1;
            //create new thread
            if(pthread_create(&threadID[ID], NULL, &handlerRequest, (void
*)(&clientSocket)) == -1 ){
                //thread create fail
                ID--;
                perror("Error : Create New Thread Failed!\n");
                break;
            }
            //else ID++, next thread
            ID++;
        }
    }
```

处理时间请求：服务端接收客户端请求，并按照数据包类型返回相应的数据。

```
                receiveLen = recv(sockfd, dataReceive, BUFFER_LEN, 0);
                if(receiveLen > 0){
                    // send message to client format, got four type
                    // TIME%s/NAME%s/LIST%s/SENDID: %d, Address: %s, Port: %d,
Sent A Message To You:#%s#
                    cout << "Data Packet Receive From Client : " <<
dataReceive << endl;
                    //if is time request
                    if( !strncmp(dataReceive, "time", 4) ){
                        //time today
                        using chrono::system_clock;
                        system_clock::time_point timeToday =
system_clock::now();
                        time_t t;
                        t = system_clock::to_time_t(timeToday);
                        char time[BUFFER_LEN];
                        strcpy(time,ctime(&t));
                        //cout << time << endl;
                        sprintf(serverTime, "TIME%s",time);
                        if( send(sockfd, serverTime, sizeof(time), 0) > 0 ){
                            cout << "Time Data Has Been Sent To CLient " <<
currentID + 1 << "!"<< endl;
                        }else{
                            //send fail
                            perror("Error : Server Send Time Data Message
Failed!\n");
                            exit(errno);
                        }
                    }
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

  客户端选择连接功能时，输入 IPv4 地址和端口（由于我的学号后 4 位是 0677，‘0’不能作为一位，所以更改为 3677，‘3’我的学号第一位）

```
peter@peter-virtual-machine:~/cn$ ./client
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|               Menu                |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 7. Exit Client                    |
+-----------------------------------+
1
Enter Server IP :
192.168.47.132
Enter Server Port :
3677
Connect Server Success!
[Packet Received] The Packet Is Connection Packet
[Received Message] : Reply Message From Server!
```

```
[Connection Established] Client Address : 192.168.47.132 PORT : 46738
Reply Message Sent Successfully!
```

- 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：

```
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|               Menu                |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 3. Request Server Time            |
| 4. Request Server Name            |
| 5. Request Client List            |
| 6. Send Message To Other Client   |
| 7. Exit Client                    |
+-----------------------------------+
3
Request Server Time Success!
[Packet Received] The Packet Is Time Packet
[Received Message] : Tue Mar 15 16:37:00 2022
```

  服务端：

```
Data Packet Receive From Client : time
Time Data Has Been Sent To CLient 1!
```

- 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：

```
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|               Menu                |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 3. Request Server Time            |
| 4. Request Server Name            |
| 5. Request Client List            |
| 6. Send Message To Other Client   |
| 7. Exit Client                    |
+-----------------------------------+
4
Request Server Name Success!
[Packet Received] The Packet Is Name Packet
[Received Message] : peter-virtual-machine
```

  服务端：

```
Data Packet Receive From Client : name
Server Name Has Been Sent To CLient 1!
```

  相关的服务器的处理代码片段：

```
                    else if( !strncmp(dataReceive, "name", 4) ) { //name
                        char tempName[100];
                        bzero(tempName, 100);
                        gethostname(tempName, 100);
                        sprintf(serverName, "NAME%s", tempName);
                        //cout << tempName;
                        if(send(sockfd, serverName, strlen(serverName),0)>0){
                            cout << "Server Name Has Been Sent To CLient " <<
currentID + 1 << "!"<< endl;
                        }else{
                            //send fail
                            perror("Error : Server Send Server Name Message
Failed!\n");
                            exit(errno);
                        }
                    }
```

- 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：（当前有两个客户端连接服务端）

```
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|                Menu               |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 3. Request Server Time            |
| 4. Request Server Name            |
| 5. Request Client List            |
| 6. Send Message To Other Client   |
| 7. Exit Client                    |
+-----------------------------------+
5
Request Client List Success!
[Packet Received] The Packet Is Client List Packet
[Received Message] : ID : 1, Address : 192.168.47.132, Port : 46924

[Packet Received] The Packet Is Client List Packet
[Received Message] : ID : 2, Address : 192.168.47.132, Port : 46926
```

  服务端：

```
Data Packet Receive From Client : list
[CLient ID : 1] Data Message Has Been Sent To Client!
[CLient ID : 2] Data Message Has Been Sent To Client!
```

  相关的服务器的处理代码片段：

```
                    else if( !strncmp(dataReceive, "list", 4) ) { //list
                        //traverse the client list
                        for(int i = 0 ; i < ID ; i++){
                            bzero(dataSend, BUFFER_LEN);
                            if(isConnect[i] == 1){
                                //check if this client id is connect or not
                                sprintf(dataSend, "LISTID : %d, Address : %s,
Port : %d\n",i+1, addrList[i], portList[i]);
                                if(send(sockfd,dataSend,strlen(dataSend), 0) > 0){
                                    //send success
                                    cout << "[CLient ID : " << i+1 << "] Data
Message Has Been Sent To Client!" << endl;
                                    sleep(0.5);
                                }else{ send fail
                                    perror("Error : Server Send Client Data
Message Failed!");
                                    exit(errno);
                                }
                            }else{
                                cout << "Client " << i+1 << " Is Not Connected" <<
endl;
                            }
                        }
                    }
```

- 客户端选择发送消息功能时，两个客户端和服务端（如果有的话）显示内容截图。

  发送消息的客户端：

```
Please Enter Number(From 1 - 7) To Select Your Option :
+-----------------------------------+
|               Menu                |
+-----------------------------------+
| 1. Connect Server                 |
| 2. Close Connection               |
| 3. Request Server Time            |
| 4. Request Server Name            |
| 5. Request Client List            |
| 6. Send Message To Other Client   |
| 7. Exit Client                    |
+-----------------------------------+
6
Please Enter The ID Of Client Which You Want To Send Message.
2
Please Enter The Message That You Want To Send.
client1 send message to client 2
[Message] : #client1 send message to client 2# Send To Client 2 Success!
```

  服务器端（可选）：

```
Data Packet Receive From Client : send#2#client1 send message to client 2
Message From Client : 1 Has Been Sent To Client : 2
[Message] client1 send message to client 2
```

  接收消息的客户端：

```
[Packet Received] The Packet Is Send Message Packet
[Received Message] : ID: 1, Address: 192.168.47.132, Port: 46906 Sent A Message
To You: client1 send message to client 2
```

  相关的服务器的处理代码片段：

```
else if(!strncmp(dataReceive, "send", 4) ){ //send message to server
                        //get client id
                        bzero(dataTemp, BUFFER_LEN);
                        int j = 0,i = 0;
                        for(i = 5 ; i < BUFFER_LEN ; i++){
                            //data receive from client string is :
send#id#msg\n
                            if(dataReceive[i] != '#'){
                                dataTemp[j] = dataReceive[i];
                                j++;
                            }else{ // == '#'
                                break;
                            }
                        }
                        sendMessageID = atoi(dataTemp);
```

```
                        j = 0;
                        bzero(sendMessage, BUFFER_LEN);
                        for(i++ ; i < BUFFER_LEN ; i++){
                            if(dataReceive[i] != '\n'){
                                sendMessage[j] = dataReceive[i];
                                j++;
                            }else{ // == '\n'
                                break;
                            }
                        }
                        bzero(dataSend, BUFFER_LEN);
                        sprintf(dataSend, "SENDID: %d, Address: %s, Port: %d
Sent A Message To You: %s"
                                , currentID+1, addrList[currentID],
portList[currentID], sendMessage);
                        if( send(fdList[sendMessageID], dataSend,
strlen(dataSend), 0) > 0 ){
                            cout << "Message From Client : " << currentID+1 <<
" Has Been Sent To Client : " << sendMessageID << endl;
                            cout << "[Message] " << sendMessage << endl;
                        }else{
                            perror("Error : Server Resend Message To Client
Failed");
                            exit(errno);
                        }
                    }
                }
```

相关的客户端（发送和接收消息）处理代码片段：

```
int Client::sendMessage(){
    char msg[1014];
    char sendMsg[1024];
    int clientID;
    cout << "Please Enter The ID Of Client Which You Want To Send Message." <<
endl;
    cin >> clientID;
    printf("Please Enter The Message That You Want To Send.\n" );
    cin >> ws; //avoid program ignore cin.getline
    cin.getline(msg, sizeof(msg));
    sprintf(sendMsg, "send#%d#", clientID);
    strcat(sendMsg, msg); // send#ID#message...
    int res = send(sock, sendMsg, sizeof(sendMsg), 0);
    if(res == -1) {
        perror("Send Message To Server Failed!\n");
        return 1;
    }
```

```
        else{
          cout << "[Message] : #"<< msg << "# Send To Client " << clientID << "
Success!" << endl << endl;
          return 0;
        }
}
```

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？

  客户端不需要调用 bind 操作，虽然客户端也能选择特定的端口进行 bind，但是一旦多个客户端在本机运行的话，就会出现端口被占用的问题。由于我们是不需要特定端口进行客户端的连接的，所以不需要调用 bind 操作，直接由操作系统分配一个端口。在每一次的连接（connect()）的时候，内核都会随机分配一个端口来连接，所以客户端的端口不会都保持不变。

- 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？

  不能连接成功，调用 listen()是为了让服务端开启监听模式，而调用 accept()是为了让服务端等待客户端的请求连接，一般都是阻塞状态。如果在 listen 和 accept 直接设置一个调试断点，那么客户端此时调用 connect 是无法得到客户端的 file description(fd)的，因此连接无法建立。

- 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？

  本次实验采用了多线程的方式来实现多个客户端的同步，每次有一个新的客户端连接服务端时就会产生新的线程，然而每个客户端的 IP 地址和端口都是唯一的，通过数组保存信息。服务端在同一个接口接收了多个客户端的数据包时，可以通过 socket 的 IP 和 Port 信息来区分数据包是哪个客户的。

- 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）

保持连接时状态：

```
peter@peter-virtual-machine:~$ netstat -an | grep 3677
tcp        0      0 0.0.0.0:3677            0.0.0.0:*               LISTEN
tcp        0      0 192.168.47.132:46926    192.168.47.132:3677     ESTABLISHED
tcp        0      0 192.168.47.132:3677     192.168.47.132:46926    ESTABLISHED
tcp        0      0 192.168.47.132:3677     192.168.47.132:46924    ESTABLISHED
tcp        0      0 192.168.47.132:46924    192.168.47.132:3677     ESTABLISHED
```

其中一个客户端断开连接后并关闭，服务端进入 CLOSE_WAIT 状态，

```
peter@peter-virtual-machine:~$ netstat -an | grep 3677
tcp        0      0 0.0.0.0:3677            0.0.0.0:*               LISTEN
tcp        0      0 192.168.47.132:46926    192.168.47.132:3677     ESTABLISHED
tcp        0      0 192.168.47.132:3677     192.168.47.132:46926    ESTABLISHED
tcp        0      0 192.168.47.132:3677     192.168.47.132:46924    CLOSE_WAIT
tcp        0      0 192.168.47.132:46924    192.168.47.132:3677     FIN_WAIT2
```

等待 60 秒左右将不再看到该客户端状态。

```
peter@peter-virtual-machine:~$ netstat -an | grep 3677
tcp        0      0 0.0.0.0:3677            0.0.0.0:*               LISTEN
```

- 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？

客户端断网后异常退出，Server 处于 CLOSE_WAIT 状态，Client 处于 FIN_WAIT2 状态。经过一段时间后，服务端变成 CLOSE_WAIT 状态，客户端状态消失。

七、 讨论、心得

本次实验是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，在实验过程中实现客户端和服务端程序还是有一定的难度。通过本次实验，我学会了基本的 Socket 编程和线程创建和处理机制，对多线程处理有了更多的了解。在实验过程中还遇到了输入消息的问题，cin.getline(msg) 输入消息一直无法获取完整的消息，花费了我很多时间来处理这个问题，最后找到问题所在然后添加一个 cin>>ws 就能解决输入的问题了。这次的实验收获了很多，通过对实验中的问题进行思考来让我们更加深入地了解服务端与客户端的连接或断开工作过程，对整个实验都基本熟悉了。

参考链接

https://www.cnblogs.com/liushao/p/6375377.html

https://www.cnblogs.com/liushui-sky/p/5609535.html

https://stackoverflow.com/questions/22008864/how-to-use-getaddrinfo-on-windows

https://docs.microsoft.com/en-us/windows/win32/api/ws2def/ns-ws2def-addrinfoa

https://zhuanlan.zhihu.com/p/365478112