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：	李文博
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：	3180104939
指导教师：	黄正谦

2021 年　1　月　　12　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：王呈　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**

以下实验记录均需结合屏幕截图（截取源代码或运行结果），进行文字标注（看完请删除本句）。

- 描述请求数据包的格式（画图说明），请求类型的定义

Length/4B	Type/4B
From/4B	To/4B
Data/0-1008B	

Length 为整个数据包的长度，最小 16Bytes；

Type 是数据包类型，具体有以下几种：

```
#define PACKTYPE_GETTIME     (3)
#define PACKTYPE_GETNAME     (4)
#define PACKTYPE_GETCLIENT   (5)
#define PACKTYPE_SENDMSG     (6)
#define PACKTYPE_ACKTIME     (7)
#define PACKTYPE_ACKNAME     (8)
#define PACKTYPE_ACKCLIENT   (9)
#define PACKTYPE_DISCONNECT  (10)
```

From 和 To 是双方 IP 地址；

Data 是数据内容，可选。

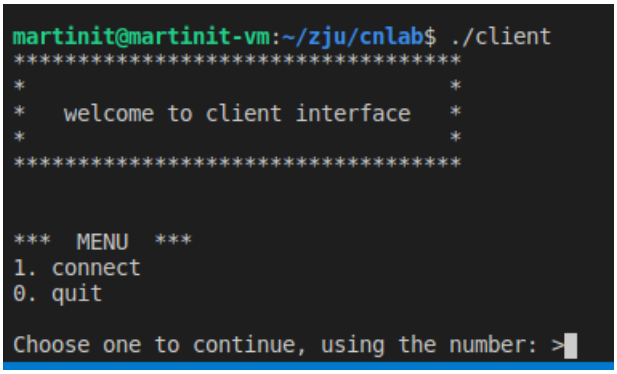
- 描述响应数据包的格式（画图说明），响应类型的定义

  同上

- 描述指示数据包的格式（画图说明），指示类型的定义

  同上

- 客户端初始运行后显示的菜单选项



- 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）

```
int main(){
    init();
    welcome();
    while(1){
        int choice=menu();
        switch(choice){
            case CONNECT: connect(); break;
            case DISCONNECT: disconnect(0); break;
            case GETTIME: gettime(); break;
            case GETNAME: getname(); break;
            case GETCLIENT: getclient(); break;
            case SENDMSG: sendmsg(); break;
            case QUIT:
                if(connected) disconnect(0);
                cout << endl << "Bye!" << endl << endl;
                return 0;
            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）

```
void child_recv_pack(){
    static vector<uchar> dat;
    uchar buf[BUF_SIZE];
    while(1){
        memset(buf, 0, sizeof(buf));
        ssize_t bytes=recv(client_sock, buf, sizeof(buf), 0);
        if(bytes>0) vector_push(dat, buf, bytes);
        uint len=is_full_pack(dat);
        if(len>=0) print_pack(dat, len);
    }
}
```

- 服务器初始运行后显示的界面

```
martinit@ubuntu-martinit:~/zju/cnlab$ ./server
Server initializing......Done
Server 192.168.31.234:4939 is running......
Waiting for new clients......
```

- 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）

```
int main(){
    init();
    while(1){
        struct sockaddr_in client_sockaddr;
        socklen_t socklen=sizeof(client_sockaddr);
        memset(&client_sockaddr, 0, sizeof(client_sockaddr));
        int client_sock=-1;
        while(client_sock==-1){
            cout << "Waiting for new clients......" << endl;
            client_sock=accept(server_sock, (struct sockaddr*)&client_
        }
        cout << "Connection to " << inet_ntoa(client_sockaddr.sin_addr
        pid_t pid=fork();
        if(pid==0){
            signal(SIGINT, child_sig_handle);
            struct client cli;
            cli.pid=pid;
            cli.sock=client_sock;
            cli.sockaddr=client_sockaddr;
            int shmid=shmget(SHM_KEY, SHM_SIZE, 0666);
            if(shmid==-1){
                cerr << "Child cannot get shared memory." << endl;
                exit(1);
            }
            child(shmid, client_sock, cli);
            exit(0);
        }else if(pid>0){
            struct client cli;
            cli.pid=pid;
            cli.sock=client_sock;
            cli.sockaddr=client_sockaddr;
            shm_add_client(shmaddr, cli);
        }else error_exit("Cannot create a child process.");
    }
}
```

- 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）

```
void child(int shmid, int client_sock, struct client cli){
    uint packlen;
    uchar* pack=get_pack(&packlen, PACKTYPE_SENDMSG, server_sockaddr.s
    send(client_sock, pack, packlen, 0);

    static vector<uchar> dat;
    uchar buf[BUF_SIZE];
    while(1){
        memset(buf, 0, sizeof(buf));
        ssize_t bytes;
        while((bytes=recv(client_sock, buf, sizeof(buf), 0))==-1);
        if(bytes>0) vector_push(dat, buf, bytes);
        uint len=is_full_pack(dat);
        if(len>=0){
            uchar *tmp=(uchar*)malloc(sizeof(uchar)*len);
            for(int i=0;i<len;i++) tmp[i]=dat[i];
            process_pack(shmid, client_sock, cli, tmp, len);
            free(tmp);
            int remain=dat.size()-len;
            tmp=(uchar*)malloc(sizeof(uchar)*remain);
            for(int i=len;i<dat.size();i++) tmp[i-len]=dat[i];
            dat.clear();
            vector_push(dat, tmp, remain);
            free(tmp);
        }
    }
}
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

```
Choose one to continue, using the number: >1
Please enter server IP address and port like '[IP]:[PORT]'
                (use port greater than 1023 for safety): >192.168.31.234:4939
Connecting 192.168.31.234:4939 ......Done

***  MENU  ***
1. connect
2. disconnect
3. get time
4. get name
5. get client list
6. send message
0. quit

Choose one to continue, using the number: >
[MESSAGE] From server:
        Hello
```

```
martinit@ubuntu-martinit:~/zju/cnlab$ ./server
Server initializing......Done
Server 192.168.31.234:4939 is running......
Waiting for new clients......
Connection to 192.168.31.101:52110 established.
Waiting for new clients......
```

Wireshark 抓取的数据包截图：

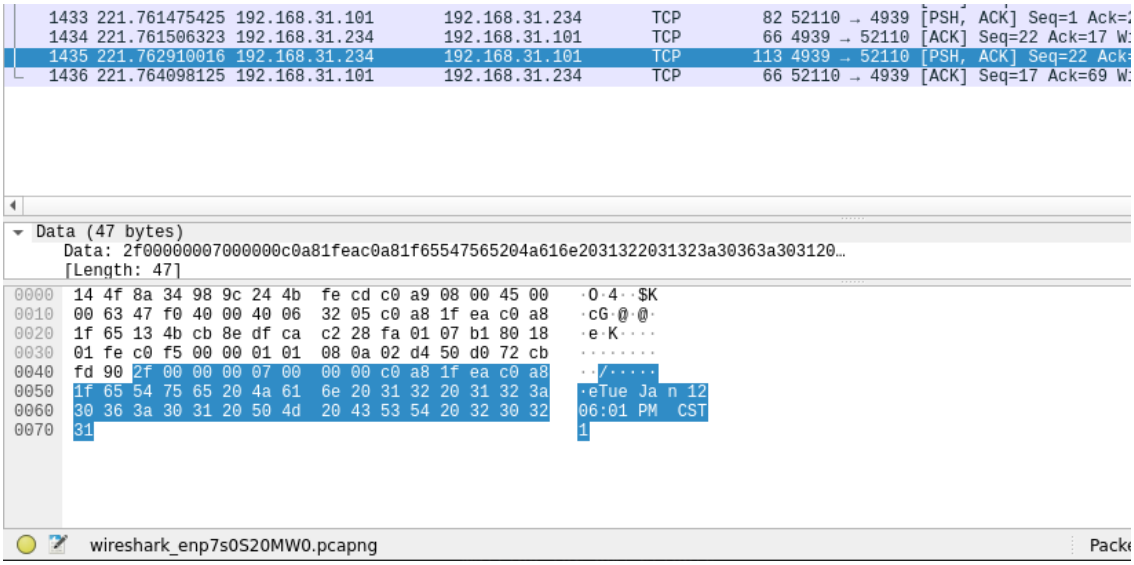
No.	Time	Source	Destination	Protocol	Length	Info
627	97.163436415	192.168.31.101	192.168.31.234	TCP	74	52110 → 4939 [SYN] Seq=0 Win=642
628	97.163473735	192.168.31.234	192.168.31.101	TCP	74	4939 → 52110 [SYN, ACK] Seq=0 Ac
629	97.164812017	192.168.31.101	192.168.31.234	TCP	66	52110 → 4939 [ACK] Seq=1 Ack=1 W
630	97.165045856	192.168.31.234	192.168.31.101	TCP	87	4939 → 52110 [PSH, ACK] Seq=1 Ac
631	97.165692870	192.168.31.101	192.168.31.234	TCP	66	52110 → 4939 [ACK] Seq=1 Ack=22

- 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。

```
Choose one to continue, using the number: >3
Sending 16 bytes to server to get time......Done

***  MENU  ***
1. connect
2. disconnect
3. get time
4. get name
5. get client list
6. send message
0. quit

Choose one to continue, using the number: >
[ACK] Time from server:
      Tue Jan 12 12:06:01 PM CST 2021
```

```
Server initializing......Done
Server 192.168.31.234:4939 is running......
Waiting for new clients......
Connection to 192.168.31.101:52110 established.
Waiting for new clients......
Sending 47 bytes time message to 192.168.31.101......Done
```

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。

```
4
Sending 16 bytes to server to get name......Done

***  MENU  ***
1. connect
2. disconnect
3. get time
4. get name
5. get client list
6. send message
0. quit

Choose one to continue, using the number: >
[ACK] Name from server:
      HOST SERVER
```

```
martinit@ubuntu-martinit:~/zju/cnlab$ ./server
Server initializing......Done
Server 192.168.31.234:4939 is running......
Waiting for new clients......
Connection to 192.168.31.101:52110 established.
Waiting for new clients......
Sending 47 bytes time message to 192.168.31.101......Done
Sending 27 bytes name message to 192.168.31.101......Done
```

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）：

```
2359 417.987780119 192.168.31.101   192.168.31.234   TCP   82 52110 → 4939 [PSH, ACK] Seq=17 Ack=69 Win=
2360 417.987810777 192.168.31.234   192.168.31.101   TCP   66 4939 → 52110 [ACK] Seq=69 Ack=33 Win=65280
2361 417.987885557 192.168.31.234   192.168.31.101   TCP   93 4939 → 52110 [PSH, ACK] Seq=69 Ack=33 Win=
2362 417.989811406 192.168.31.101   192.168.31.234   TCP   66 52110 → 4939 [ACK] Seq=33 Ack=96 Win=64256
```

```
▸ Frame 2361: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface enp7s0, id 0
▸ Ethernet II, Src: ASUSTekC_cd:c0:a9 (24:4b:fe:cd:c0:a9), Dst: IntelCor_34:98:9c (14:4f:8a:34:98:9c)
▸ Internet Protocol Version 4, Src: 192.168.31.234, Dst: 192.168.31.101
0000  14 4f 8a 34 98 9c 24 4b  fe cd c0 a9 08 00 45 00   ·O·4··$K ····E·
0010  00 4f 47 f2 40 00 40 06  32 17 c0 a8 1f ea c0 a8   ·OG·@·@· 2·····
0020  1f 65 13 4b cb 8e df ca  c2 57 fa 01 07 c1 80 18   ·e·K···· ·W····
0030  01 fe c0 e1 00 00 01 01  08 0a 02 d7 4f 51 72 ce   ········ ····OQr·
0040  fc 14 1b 00 00 00 08 00  00 00 c0 a8 1f ea c0 a8   ········ ········
0050  1f 65 48 4f 53 54 20 53  45 52 56 45 52            ·eHOST S ERVER
```

相关的服务器的处理代码片段：

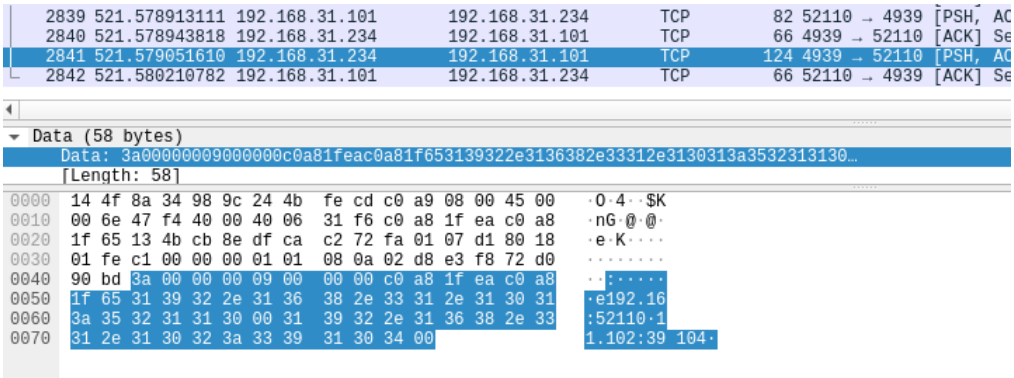
```
case PACKTYPE_GETNAME:{
    uint packlen;
    uchar* pack=get_pack(&packlen, PACKTYPE_ACKNAME, server_sockaddr.sin_addr.s_addr, c
    cout << "Sending " << packlen << " bytes name message to " << inet_ntoa(cli.sockadd
    if(send(client_sock, pack, packlen, 0)==packlen) cout << "Done" << endl;
    else cout << "Failed" << endl;
    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。

```
5
Sending 16 bytes to server to get client list......Done

***  MENU  ***
1. connect
2. disconnect
3. get time
4. get name
5. get client list
6. send message
0. quit

Choose one to continue, using the number: >
[ACK] Client list from server:
      No.    IP
      1      192.168.31.101:52110
      2      192.168.31.102:39104
```

```
martinit@ubuntu-martinit:~/zju/cnlab$ ./server
Server initializing......Done
Server 192.168.31.234:4939 is running......
Waiting for new clients......
Connection to 192.168.31.101:52110 established.
Waiting for new clients......
Sending 47 bytes time message to 192.168.31.101......Done
Sending 27 bytes name message to 192.168.31.101......Done
Connection to 192.168.31.102:39104 established.
Waiting for new clients......
Sending 58 bytes clients message to 192.168.31.101......Done
```

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：

```
case PACKTYPE_GETCLIENT:{
    uint content_len;
    uchar* content=write_client_list(&content_len, shm_get_clients(shmaddr));
    uint packlen;
    uchar* pack=get_pack(&packlen, PACKTYPE_ACKCLIENT, server_sockaddr.sin_addr.s_addr,
    cout << "Sending " << packlen << " bytes clients message to " << inet_ntoa(cli.sock
    if(send(client_sock, pack, packlen, 0)==packlen) cout << "Done" << endl;
    else cout << "Failed" << endl;
    break;
}
case PACKTYPE_SENDMSG:{
```

- 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：

```
6
Please enter destination IP: >192.168.31.102
Enter the message you want to send, end with ENTER: >who are you?
Sending 28 bytes message to 192.168.31.102......Done

***  MENU  ***
1. connect
2. disconnect
3. get time
4. get name
5. get client list
6. send message
0. quit

Choose one to continue, using the number: >
[MESSAGE] From server:
          Successfully send message to 192.168.31.102
```

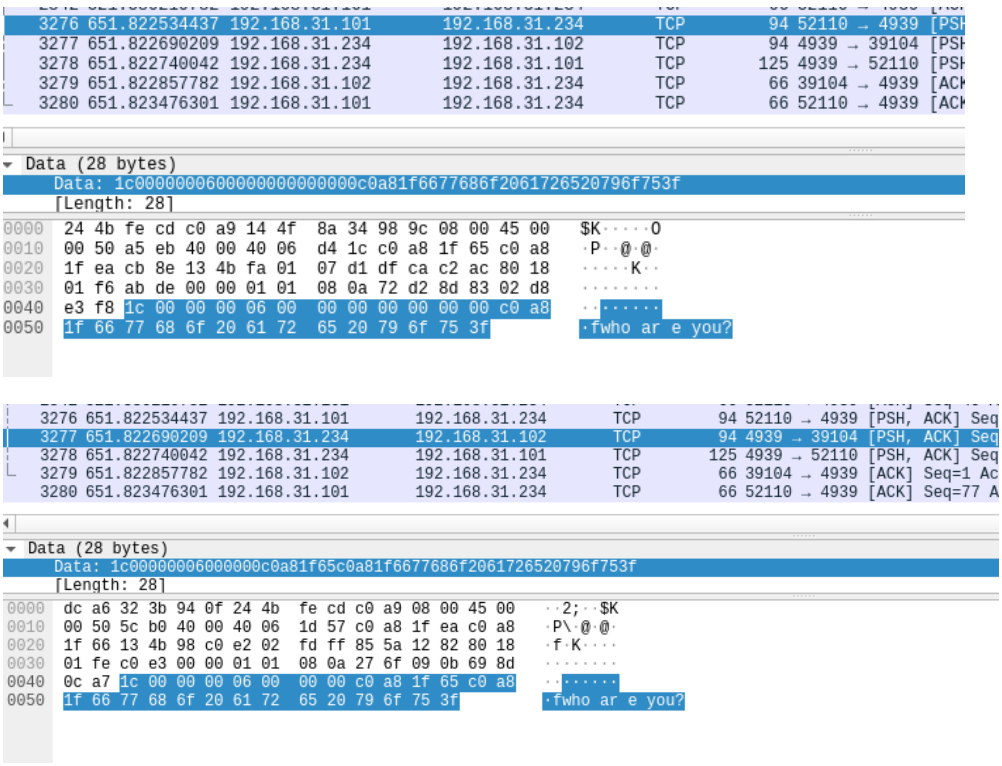
服务器：

```
Transponding 28 bytes message from 192.168.31.101 to 192.168.31.102......Done
Waiting for new clients......
Noticing 192.168.31.101......Done
```

接收消息的客户端：

```
[MESSAGE] From 192.168.31.101:
          who are you?
```

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 PACKTYPE_SENDMSG:{
    in_addr_t to=*(in_addr_t*)(pack+TO_OFFSET);
    if(to==server_sockaddr.sin_addr.s_addr){
        cout << endl << "[MESSAGE] From " << inet_ntoa(cli.
        cout << "          ";
        for(int j=CONTENT_OFFSET;j<packlen;j++) cout << pac
        cout << endl;
        break;
    }else{
        int pos=shm_find_ip(shmaddr, to);
        if(pos==-1){
            cout << "Received a message package with an unc
            string content="Destination ";
            content+=inet_ntoa(to);
            content+=" does not exist!!!";
            uint packlen;
            uchar* pack=get_pack(&packlen, PACKTYPE_SENDMSG
            cout << "Noticing " << inet_ntoa(cli.sockaddr.s
            if(send(client_sock, pack, packlen, 0)==packlen
            else cout << "Failed" << endl;
        }else{
            in_addr_t *pfrom=(in_addr_t*)(pack+FROM_OFFSET)
            *pfrom=cli.sockaddr.sin_addr.s_addr;
            cout << "Transponding " << packlen << " bytes m
            uchar *tmp=(uchar*)shmaddr+SHM1_SIZE;
            memset(tmp, 0, SHM_TMP_SIZE);
            *(uint*)tmp=packlen;
            tmp+=sizeof(packlen);
            for(int i=0;i<packlen;i++) tmp[i]=pack[i];
            kill(getppid(), SIGUSR1);
            while(((uchar*)shmaddr)[SHM_SIZE-1]==0) usleep(
            int ret=((uchar*)shmaddr)[SHM_SIZE-1];
            memset((uchar*)shmaddr+SHM1_SIZE, 0, SHM_TMP_SI
            if(ret==packlen){
                cout << "Done" << endl;
                string content="Successfully send message t
                content+=inet_ntoa(to);
                uint packlen;
                uchar* pack=get_pack(&packlen, PACKTYPE_SEN
```

相关的客户端（发送和接收消息）处理代码片段：

```
witch(type){
    case PACKTYPE_SENDMSG:{
        cout << endl << "[MESSAGE] From " << (from==server_sockadd
        cout << "          ";
        for(int j=k;j<len;j++) cout << dat[j];
        cout << endl;
        break;
    }
    case PACKTYPE_ACKTIME:{
        cout << endl << "[ACK] Time from server:" << endl;
        cout << "      ";
        for(int j=k;j<len;j++) cout << dat[j];
        cout << endl;
        break;
    }
    case PACKTYPE_ACKNAME:{
        cout << endl << "[ACK] Name from server:" << endl;
        cout << "      ";
        for(int j=k;j<len;j++) cout << dat[j];
        cout << endl;
        break;
    }
    case PACKTYPE_ACKCLIENT:{
        cout << endl << "[ACK] Client list from server:" << endl;
        print_client_list(dat, k, len, 6);
        cout << 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。

  TCP 连接已断开，客户端没有发出 TCP 释放消息

  没有变化。

- 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？

  client 退出时会发出 DISCONNECT 的数据包，如果很快就连上网线，TCP 未超时的情况下 server 是可以收到 disconnect 消息的，client 列表也就不会有之前的这个 client。如果等待较长时间，TCP 超时，server 收不到 DISCONNECT 的时候，之前的异常 client 还会存在，给他发消息会失败，因为 ip:port 已经改变，之前的 socket 失效了。

- 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。

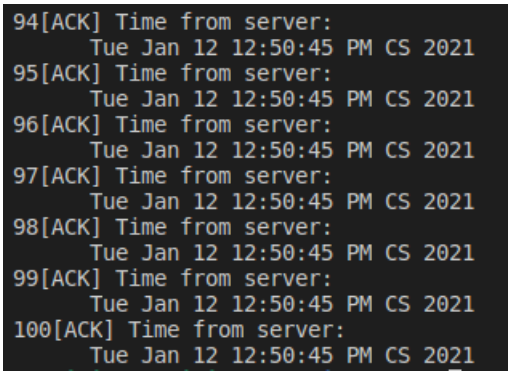
```
93[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
94[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
95[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
96[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
97[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
98[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
99[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
100[ACK] Time from server:
      Tue Jan 12 12:48:53 PM CS 2021
```

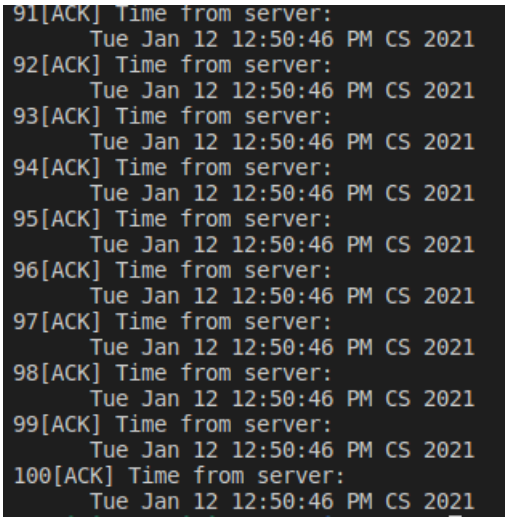
```
92Sending 47 bytes time message to 192.168.31.101......Done
93Sending 47 bytes time message to 192.168.31.101......Done
94Sending 47 bytes time message to 192.168.31.101......Done
95Sending 47 bytes time message to 192.168.31.101......Done
96Sending 47 bytes time message to 192.168.31.101......Done
97Sending 47 bytes time message to 192.168.31.101......Done
98Sending 47 bytes time message to 192.168.31.101......Done
99Sending 47 bytes time message to 192.168.31.101......Done
100Sending 47 bytes time message to 192.168.31.101......Done
```

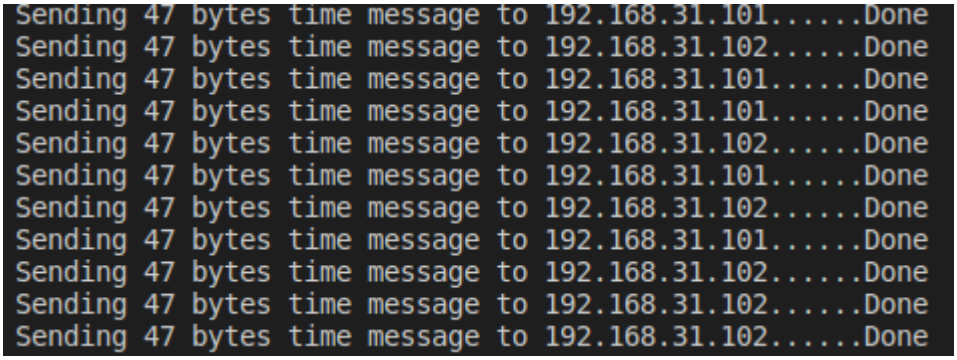
```
9371 1954.8193180… 192.168.31.101   192.168.31.234   TCP   74 52170 → 493
9372 1954.8193492… 192.168.31.234   192.168.31.101   TCP   74 4939 → 5217
9373 1954.8207707… 192.168.31.101   192.168.31.234   TCP   66 52170 → 493
9374 1954.8210434… 192.168.31.234   192.168.31.101   TCP   87 4939 → 5217
9375 1954.8218317… 192.168.31.101   192.168.31.234   TCP   66 52170 → 493
9376 1958.1624961… 192.168.31.101   192.168.31.234   TCP   82 52170 → 493
9377 1958.1625250… 192.168.31.234   192.168.31.101   TCP   66 4939 → 5217
9378 1958.1624964… 192.168.31.101   192.168.31.234   TCP 1514 52170 → 493
9379 1958.1625352… 192.168.31.234   192.168.31.101   TCP   66 4939 → 5217
9380 1958.1633979… 192.168.31.101   192.168.31.234   TCP  202 52170 → 493
9381 1958.1634054… 192.168.31.234   192.168.31.101   TCP   66 4939 → 5217
9382 1958.1639808… 192.168.31.234   192.168.31.101   TCP  113 4939 → 5217
9383 1958.1647489… 192.168.31.101   192.168.31.234   TCP   66 52170 → 493
9384 1958.1651842… 192.168.31.234   192.168.31.101   TCP  113 4939 → 5217
9385 1958.1659735… 192.168.31.101   192.168.31.234   TCP   66 52170 → 493

Data: 10000000030000000000000c0a81fea10000000030000000000000c0a81fea10000000…

0040  a7 af 10 00 00 00 03 00  00 00 00 00 00 c0 a8
0050  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
0060  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
0070  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
0080  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
0090  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
00a0  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8
00b0  1f ea 10 00 00 00 03 00  00 00 00 00 00 c0 a8

Data (data.data), 1,448 bytes
```

只能抓到几十个包。

- 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







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，系统会自动选择使用的 ip:port。

  不是。

- 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，listen 只是设置了监听状态，要调用 accept 才会真正监听。

- 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？

  每一个接收线程对应一个 client,不会冲突,而且每个线程只会使用对应的 client 的 socket 去接收，而且其他 socket 在这个进程内是无效的。

- 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）

  TIME_WAIT，大约一到两分钟。

- 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？

  没变化，因为 client 没有通知 server 断开连接。
  给 client 发一个包来检测，但肯定无效了。

七、 讨论、心得

socket 使用并不难，本实验主要是多线程编程和他们之间的同步比较麻烦，所以在实验之后，由 windows 转向了 linux，因为 linux 的多线程和 socket 函数都比较简单，但 linux 机器比较稀缺，所以本次实验使用的是 linux 台式机（server）+windows 笔记本上的 linux 虚拟机（client1）+树莓派（client2）来完成的。