

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：

2018 年 03 月 24 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

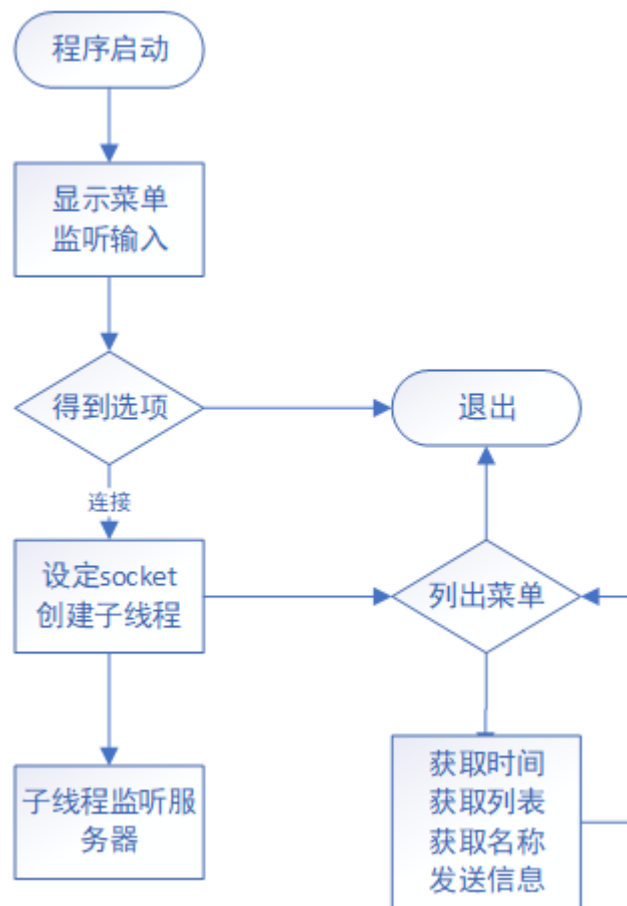
装通过调用 `send()` 发给接收客户端（使用接收客户端的 `socket` 句柄）。

- 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
- 客户端和服务端框架图（用流程图表示）



- 客户端初始运行后显示的菜单选项

```
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket
File Edit View Search Terminal Help
zzx@zzx-VM ~/Desktop $ cd ../Documents/Sockets/hw/Networking_P2_Socket/
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket $ ./tcp_client
Welcome to network experiment2!
1. Connect to specific IP and port.
7. Exit.
Please enter the command:
1
Please enter ip:
127.0.0.1
Please enter port:
5330
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!
```

我们看到，当初始运行的时候， 只有两个选项可选，就是连接到服务器或退出。

- 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 *ListenResponse(void * arg)
{
    int network_socket;
    char server_response[256];
    network_socket = (int)(*(int*)arg);
    memset(server_response, 0, sizeof(server_response));
    while (1)
    {
        Int recv_ret =
        recv(network_socket,&server_response,sizeof(server_response), 0);
        if (recv_ret > 0)
        {
            printf("[Data received] ");
            printf("%s\n", server_response);
            memset(server_response, 0, sizeof(server_response));
        }
        else if (recv_ret < 0)
        {
            perror("Invalid socket\n");
            pthread_exit(NULL);
        }
        else if (recv_ret == 0)
```

```

        {
            printf("Socket closed, exiting current thread\n");
            pthread_exit(NULL);
        }
    }
    pthread_exit(NULL);
}

```

使用 `recv` 阻塞接受服务器发来的信息，收到之后进行输出处理，整个逻辑使用 `while(1)` 包装，当接受不到消息 `recv` 的返回值为 0 或负数，即说明连接断开，退出子线程。

- 服务器初始运行后显示的界面

(1) 服务端初始化运行：

```

laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success

```

(2) 客户端连接服务端：

```

laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 192.168.52.151: 59150
data send success!

```

(3) 客户端发送请求后，服务端回复请求：

```

laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 192.168.52.151: 59150
data send success!
data receive: timedata send success!
data receive: namedata send success!

```

- 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）

- 每一个连接的客户端都使用 `pthread_create()` 函数创建子线程：基本思路为每次连接一个客户端都创建一个新的线程

```
// communicate
```

```

while(1){
    // accept
    length = sizeof(struct sockaddr);
    newfd = accept(sockfd, (struct sockaddr *)&c_addr, &length);
    if(newfd == -1){
        perror("accept error!\n");
        continue;
    }
    else{
        // connect
        printf("A new connection occurs!\n");
        printf("The client is: %s: %d \n", inet_ntoa(c_addr.sin_addr),
ntohs(c_addr.sin_port));
        // store address and port
        alladdr[id] = (char*)malloc(sizeof(char) * ADDR_LENGTH);
        strcpy(alladdr[id], inet_ntoa(c_addr.sin_addr));
        allport[id] = ntohs(c_addr.sin_port);
        // set flag and status as connected
        isConnect[id] = 1;
        // create new thread
        if(pthread_create(&thread_id[id], NULL, (void *)&communicate),(void
*)(&newfd)) == -1){
            id--;
            perror("thread create error!\n");
            break;
        }
        id++;
    }
}

```

• 获取时间:

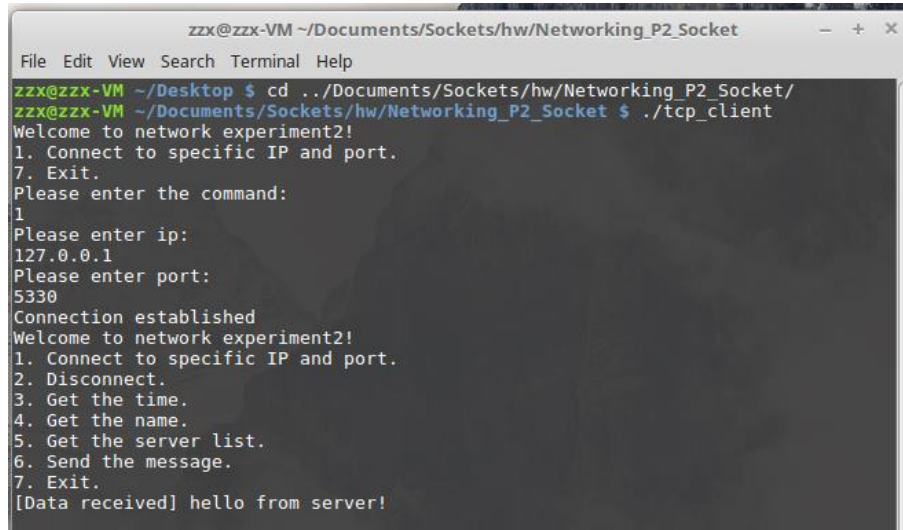
```

// time()
if(!strcmp(data_recv, "time", 4)){
    totalsec = (int)time(0);
    sec = totalsec % 60;
    min = totalsec % 3600 / 60;
    hour = (totalsec % (24 * 60) / 3600 + 8) % 24;
    sprintf(localtime, "%02d:%02d:%02d\n", hour, min, sec);
    // send time to client
    if( send(newfd, localtime, strlen(localtime), 0) > 0){
        printf("data send success!\n");
    }
    else{
        perror("data send: Server has NOT sent your message!\n");
        exit(errno);
    }
}

```

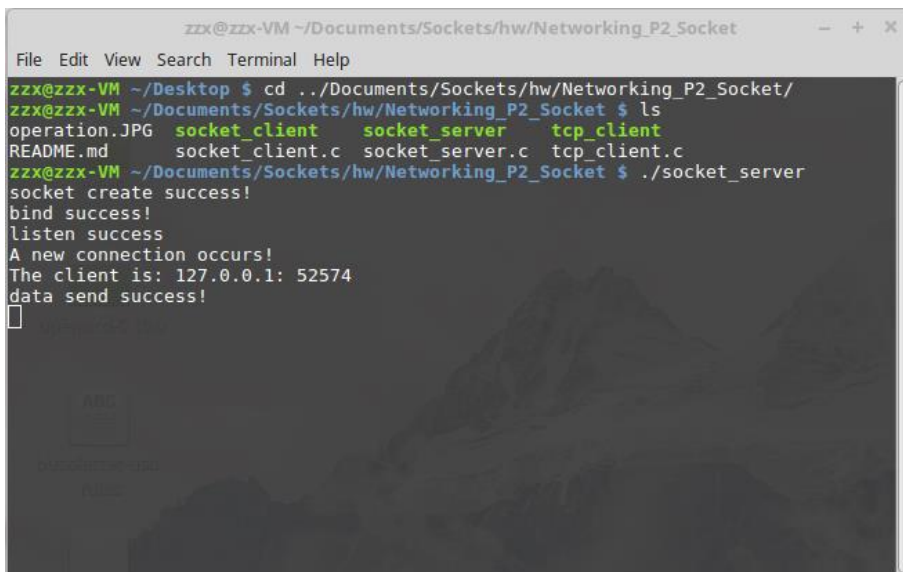
```
}  
}
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。



```
zzx@zzx-VM ~/Documents/Socket/hw/Networking_P2_Socket  
File Edit View Search Terminal Help  
zzx@zzx-VM ~/Desktop $ cd ../Documents/Socket/hw/Networking_P2_Socket/  
zzx@zzx-VM ~/Documents/Socket/hw/Networking_P2_Socket $ ./tcp_client  
Welcome to network experiment2!  
1. Connect to specific IP and port.  
7. Exit.  
Please enter the command:  
1  
Please enter ip:  
127.0.0.1  
Please enter port:  
5330  
Connection established  
Welcome to network experiment2!  
1. Connect to specific IP and port.  
2. Disconnect.  
3. Get the time.  
4. Get the name.  
5. Get the server list.  
6. Send the message.  
7. Exit.  
[Data received] hello from server!
```

客户端



```
zzx@zzx-VM ~/Documents/Socket/hw/Networking_P2_Socket  
File Edit View Search Terminal Help  
zzx@zzx-VM ~/Desktop $ cd ../Documents/Socket/hw/Networking_P2_Socket/  
zzx@zzx-VM ~/Documents/Socket/hw/Networking_P2_Socket $ ls  
operation.JPG  socket_client  socket_server  tcp_client  
README.md     socket_client.c socket_server.c tcp_client.c  
zzx@zzx-VM ~/Documents/Socket/hw/Networking_P2_Socket $ ./socket_server  
socket create success!  
bind success!  
listen success  
A new connection occurs!  
The client is: 127.0.0.1: 52574  
data send success!  
[ ]
```

服务端

可以看到，当有一个新的连接的时候，服务端会进行检测，并显示出连接的客户端详情，ip 地址是 127.0.0.1 端口是 52574。同时，我们知道，一开始服务端将发送一个 hello 信号，在客户端上，可以看到这样的消息。而在服务端上也有发送成功的提示。

- 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。


```
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket
File Edit View Search Terminal Help
Welcome to network experiment2!
1. Connect to specific IP and port.
7. Exit.
Please enter the command:
1
Please enter ip:
127.0.0.1
Please enter port:
5330
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!
3
[Data received] 08:24:04
```

客户端

```
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket
File Edit View Search Terminal Help
zzx@zzx-VM ~/Desktop $ cd ../Documents/Sockets/hw/Networking_P2_Socket/
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket $ ls
operation.JPG  socket_client  socket_server  tcp_client
README.md     socket_client.c socket_server.c tcp_client.c
zzx@zzx-VM ~/Documents/Sockets/hw/Networking_P2_Socket $ ./socket_server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 127.0.0.1: 52574
data send success!
[data receive]: time
data send success!
```

服务端

可以看到，客户端发出 3 指令的时候，服务端收到一条 time 请求，于是给客户端发去了当前的时间，而客户端也收到了当前时间。

- 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。
 - 服务端：

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 192.168.52.151: 46580
data send success!
A new connection occurs!
The client is: 192.168.52.151: 46582
data send success!
data receive: namedata send success!

```

• 客户端:

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
Welcome to network experiment2!
1. Connect to specific IP and port.
7. Exit.
Please enter the command:
1
Please enter ip:
192.168.52.151
Please enter port:
5330
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!
4
[Data received] ubuntu

```

相关的服务器的处理代码片段：该代码主要用来获取名字信息

```

// name()
else if(!strncmp(data_recv, "name", 4)){
    // get host name
    gethostname(hostname, sizeof(hostname));
    sprintf(hostname, "%s\n", hostname);
    // send time to client
    if( send(newfd, hostname, strlen(hostname), 0) > 0){
        printf("data send success!\n");
    }
    else{
        perror("data send: Server has NOT sent your message!\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。

- 服务端：

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 192.168.52.151: 46580
data send success!
A new connection occurs!
The client is: 192.168.52.151: 46582
data send success!
data receive: namedata send success!
data receive: listdata send success!
data send success!

```

- 客户端：

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
1
Please enter ip:
192.168.52.151
Please enter port:
5330
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!

4
[Data received] ubuntu

5
[Data received] id: 1, addr: 192.168.52.151, port: 46580
id: 2, addr: 192.168.52.151, port: 46582

```

相关的服务器的处理代码片段：

```

// list()

if(!strcmp(data_recv, "list", 4)){

```

```

// traverse the client list
for(i = 0; i < id; i++){
    bzero(data_send, BUFFER_LENGTH);           // initialize
    // test if is still connected
    if(isConnect[i] == 1){
        sprintf(data_send, "id: %d, addr: %s, port: %d\n", i+1,
alladdr[i], allport[i]);

        if( send(newfd, data_send, strlen(data_send), 0) > 0){
            printf("data send success!\n");
        }
        else{
            error("data send: Server has NOT sent your
message!\n");

            exit(errno);
        }
    }
}
}

```

- 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：

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!

4
[Data received] ubuntu

5
[Data received] id: 1, addr: 192.168.52.151, port: 46580
id: 2, addr: 192.168.52.151, port: 46582

6
Please input the number of the client:
2
Please input the message you want to send:
hello world another me hhhhhh!

```

服务器端（可选）：

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
laylalaisyc@ubuntu:~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket$ ./socket_
server
socket create success!
bind success!
listen success
A new connection occurs!
The client is: 192.168.52.151: 46580
data send success!
A new connection occurs!
The client is: 192.168.52.151: 46582
data send success!
data receive: namedata send success!
data receive: listdata send success!
data send success!
data receive: send:2,hello world another me hhhhhhhh!
data send success!
]

```

接收消息的客户端:

```

laylalaisyc@ubuntu: ~/Desktop/mygit/ZJU_3_2_ComputerNetworks/p2_socket
Welcome to network experiment2!
1. Connect to specific IP and port.
7. Exit.
Please enter the command:
1
Please enter ip:
192.168.52.151
Please enter port:
5330
Connection established
Welcome to network experiment2!
1. Connect to specific IP and port.
2. Disconnect.
3. Get the time.
4. Get the name.
5. Get the server list.
6. Send the message.
7. Exit.
[Data received] hello from server!
[Data received] id: 1, addr: 192.168.52.151, port: 46580 send you a message:
hello world another me hhhhhhhh!

```

相关的服务器的处理代码片段: 通过输入获得将要发送信息的目的客户端, 然后从数组中获得该客户端的信息, 最后完成信息的发送

```

// client want to send: "send:58,message"
if(!strcmp(data_recv, "send", 4)){
    // get id of the receiver client
    j = 0;
    bzero(data_temp, BUFFER_LENGTH);
    for(i = 5; i < BUFFER_LENGTH; i++){
        if(data_recv[i] != ','){
            data_temp[j] = data_recv[i];

```

```

        j++;
    }
    else{
        break;
    }
}
toID = atoi(data_temp);
// get message client want to send
j = 0;
bzero(toMessage, BUFFER_LENGTH);
for(i++; i < BUFFER_LENGTH; i++){
    if(data_recv[i] != '\n'){
        toMessage[j] = data_recv[i];
        j++;
    }
    else{
        break;
    }
}
// information to send: client's information + message
bzero(data_send, BUFFER_LENGTH); // initialize
sprintf(data_send, "id: %d, addr: %s, port: %d send you a
message:\n%s\n", curID+1, alladdr[curID], allport[curID], toMessage);
if( send(allfd[toID], data_send, strlen(data_send), 0) > 0){
    printf("data send success!\n");
}
else{
    perror("data send: Server has NOT sent your message!\n");
    exit(errno);
}
}
}

```

相关的客户端（发送和接收消息）处理代码片段：

接收消息的处理代码：（子线程）

在连接时使用 `pthread` 创建一个线程，将 `socket` 传入，并调用 `recv` 监听有无消息到达。

```

void *ListenResponse(void * arg)
{
    int network_socket;
    char server_response[256];
    network_socket = (int)(*(int*)arg);
    memset(server_response, 0, sizeof(server_response));
    while (1)
    {

```

```

    int recv_ret =
recv(network_socket,&server_response,sizeof(server_response), 0);
    if (recv_ret > 0)
    {
        printf("[Data received] ");
        printf("%s\n", server_response);
        memset(server_response, 0, sizeof(server_response));
    }
    else if (recv_ret < 0)
    {
        perror("Invalid socket\n");
        pthread_exit(NULL);
    }
    else if (recv_ret == 0)
    {
        printf("Socket closed, exiting current thread\n");
        pthread_exit(NULL);
    }
}
pthread_exit(NULL);
}

```

发送消息的处理代码：（发送消息以 send 开头示意这是一个发送给特定服务器的消息）

```

int Clisendserver(int network_socket)
{
    if (network_socket == -1)
    {
        perror("Socket not available");
        return 1;
    }
    char msg[246];
    char msgno[256];
    int rv, no;
    printf("Please input the number of the client:\n");
    scanf("%d", &no);
    printf("Please input the message you want to send:\n");
    scanf("%s", msg);
    //itoa(no, msgno, 10);
    sprintf(msgno, "send:%d,", no);
    strcat(msgno, msg);
    rv = send(network_socket,msgno, sizeof(msgno), 0);
    if (rv == -1)
        return 1;
}

```

```
else
    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，虽然客户端也可以选择特定端口 bind，但是如果多个客户端在本机同时运行，则会出现端口被占用的问题，由于我们不需要特定端口进行客户端连接，所以不需要调用 bind。不调用 bind，则由操作系统内核分配一个端口号。每一次调用 connect，内核会随机分配一个端口连接，所以不会都保持不变。

- 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。Listen（）的作用是服务器开启监听模式，而 accept（）的作用是服务器等待客户端连接，一般都是阻塞态。如果在 listen 和 accept 之间设置了一个调试断点，那么当客户端调用 connect 之后虽然服务端可以通过 listen 函数接收到客户端的信息，但是由于没有执行 accept 就无法得到客户端的 file description，因此连接不能成功。

- 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？

在本次实验中采用了多线程的方式来实现多个客户端的同步，也就是说每次有一个新的客户端连接服务端的时候都产生了新的线程。而每个客户端的 ip 地址以及 port 地址是唯一的，通过一系列数组保存该信息。每次当服务器在同一个接口接收多个客户端的数据时，通过 socket 的 ip 和 port 信息可以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）。

```
zzx@zzx-VM ~/Desktop $ netstat -an | grep 5330
tcp        0      0 0.0.0.0:5330          0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:52672      127.0.0.1:5330       ESTABLISHED
tcp        0      0 127.0.0.1:5330       127.0.0.1:52672      ESTABLISHED
```

原本 Server 与 Client 处于 ESTABLISHED 状态。


```

zzx@zzx-VM ~/Desktop $ netstat -an | grep 5330
tcp        0      0 0.0.0.0:5330      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:5330    127.0.0.1:52672  TIME_WAIT

```

客户端断开连接，服务端进入 TIME_WAIT 状态。这个状态大概持续 2 倍 MSL 时长，linux 下约 60 秒左右，之后连接被关闭，状态变成 closed。

```

zzx@zzx-VM ~/Desktop $ netstat -an | grep 5330
tcp        0      0 0.0.0.0:5330      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？

Server 处于 CLOSE_WAIT，而 Client 处于 FIN_WAIT2 状态。

```

zzx@zzx-VM ~/Desktop $ netstat -an | grep 5330
tcp        0      0 0.0.0.0:5330      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:52670    127.0.0.1:5330    FIN_WAIT2
tcp        0      0 127.0.0.1:5330     127.0.0.1:52670    CLOSE_WAIT

```

经过一段时间后，服务器变成了 CLOSE_WAIT，客户端状态消失。

```

zzx@zzx-VM ~/Desktop $ netstat -an | grep 5330
tcp        0      0 0.0.0.0:5330      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:5330     127.0.0.1:52670    CLOSE_WAIT

```

retval = select(maxfd+1, &rfd, NULL, NULL, &tv); 服务器通过 select () 函数来检测客户端是否连接，如果返回为-1 的时候连接无效。

七、 讨论、心得

本次实验主要是完成 socket 的编程，其中最大的困难是实现多客户端的同步并发以及客户端之间的消息传递。虽然最初比较没有思路，但是后来通过数组存储客户端的 ip 以及 port 信息，可以较好的实现该功能。整个实验主要是在理解 socket 的基础上，完成多个功能的实现。通过本次实验，加深了我对 socket 信息传递的理解。