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：	应承峻
学　　院：	计算机学院
系：	软件工程系
专　　业：	软件工程
学　　号：	3170103456
指导教师：	高艺

2019 年 10 月 19 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：夏林轩　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

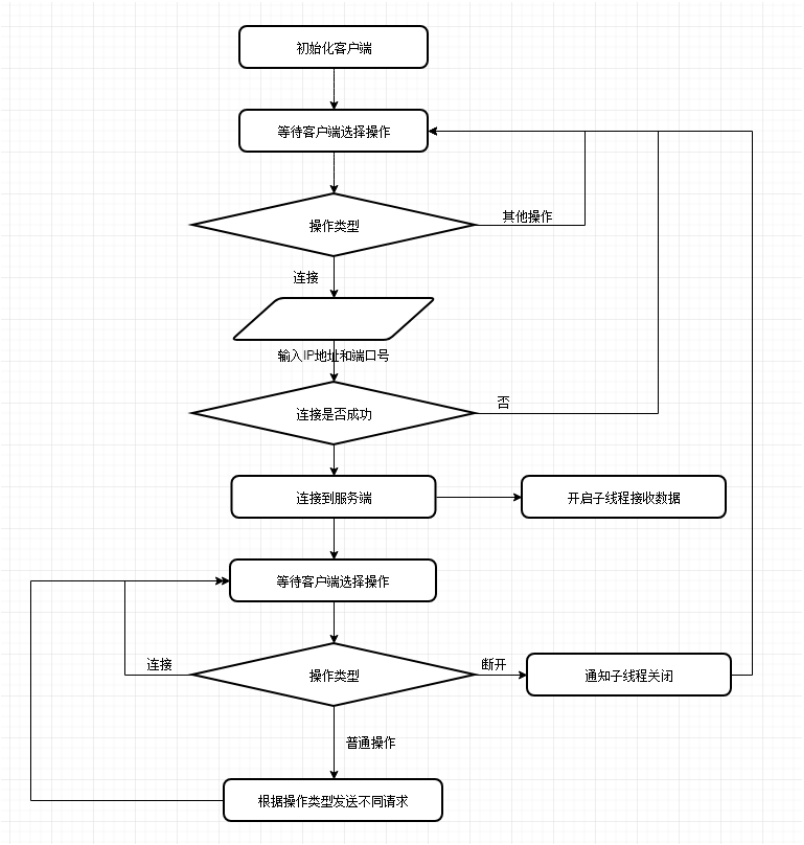
五、 实验数据记录和处理

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：

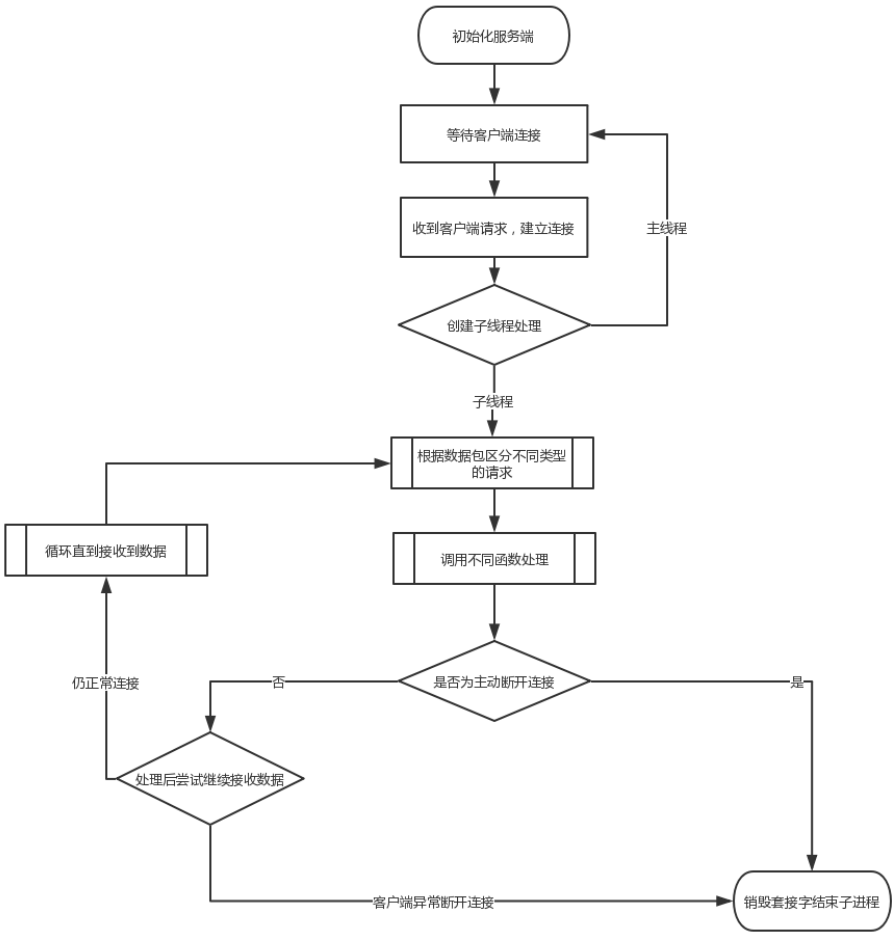
- **源代码：客户端和服务端的代码分别在一个目录**
- **可执行文件：可运行的.exe 文件或 Linux 可执行文件，客户端和服务端各一个**

- 客户端和服务器端框架图（用流程图表示）

客户端框架图：



服务端框架图：



- 客户端初始运行后显示的菜单选项

```
+------------------------------------------------+
|                    请选择                      |
+------------------------------------------------+
|[1]请求连接到指定地址和端口的服务端             |
|[2]断开与服务端的连接                           |
|[3]请求服务端给出当前时间                       |
|[4]请求服务端给出其机器的名称                   |
|[5]请求服务端给出当前连接的所有客户端信息       |
|[6]请求服务端把消息转发给对应编号的客户端       |
|[0]断开连接并退出客户端程序                     |
+------------------------------------------------+
你的选择：
```

- 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）

  在连接成功后，会创建一个子线程来执行 SocketListen：

```
cout << "连接成功!\n";
task = true;
thread listener(socketListener);
listener.detach();
```

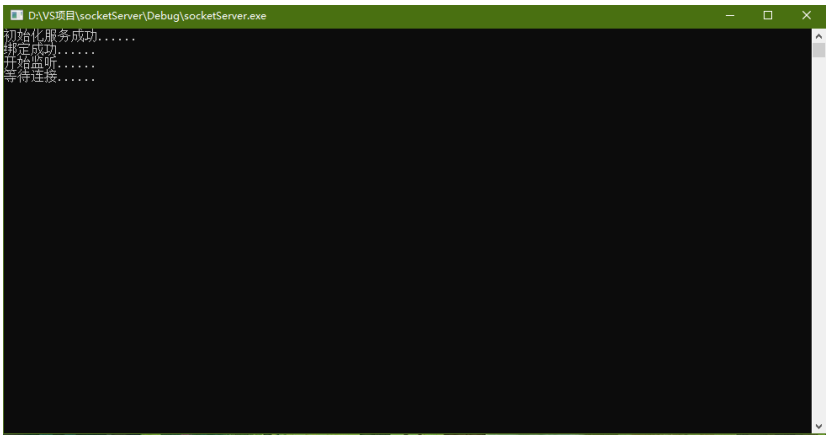
其中 task 用于控制子线程的 while 循环，只要不断开连接，while 循环始终在执行，因此可以监听服务端发送的数据包。在循环中，每次判断拿到数据包的状态 flag，并打印结果：

```
void socketListener() {
    while (task) {
        Packet p = receivePacket();
        if (p.flag == 0) {
            cout << "ERROR: 接收失败!\n";
        } else if (p.flag == 2) {
            conn.close();
            task = false;
            cout << "成功断开连接!\n";
        } else {
            cout << p.content << endl;  //normal case
        }
        Sleep(100);
    }
}
```

当客户端执行断开连接或退出后，task 的值为 0，使得子线程停止监听：

```
} else if (p.flag == 2) {
    conn.close();
    task = false;
    cout << "成功断开连接!\n";
} else {
```

- 服务器初始运行后显示的界面



- 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）

```
for (;;) {
    if (ret < 0) {
        changeClientList(client, UNCONNECTED);
        shutdown(client, SD_BOTH);
        break;
    }
    else if(ret > 0){
        ::memcpy(&packet, revData, sizeof(packet));//把接收到的信息转换成结构体
        packet.content[packet.length] = 0;
        printf("开始处理%d号客户端的请求\n", id);
        printf("length:%d  flag:%d  content: %s\n", packet.length, packet.flag, packet.content);
        switch (packet.flag){ ... }
    }
    Sleep(100);
    ret = recv(client, revData, 1024, 0);
}
```

使用一个无限的 for 循环，通过 recv()函数的返回值来判断与客户端的连接状态，如果收到数据包就进行相应处理，如果返回负值就表示连接中断，直接跳出无限循环，结束子线程。

- 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：

```
+------------------------------------------------+
|                    请选择                      |
+------------------------------------------------+
|[1]请求连接到指定地址和端口的服务端            |
|[2]断开与服务端的连接                          |
|[3]请求服务端给出当前时间                      |
|[4]请求服务端给出其机器的名称                  |
|[5]请求服务端给出当前连接的所有客户端信息      |
|[6]请求服务端把消息转发给对应编号的客户端      |
|[0]断开连接并退出客户端程序                    |
+------------------------------------------------+
你的选择：1
连接成功!
```

  服务端：

```
D:\VS项目\socketServer\Debug\socketServer.exe
初始化服务成功......
绑定成功......
开始监听......
等待连接......
收到请求来自:192.168.43.7, 客户端编号为:1
开始处理1号客户端的请求
length:22  flag:1  content: request for connection
```

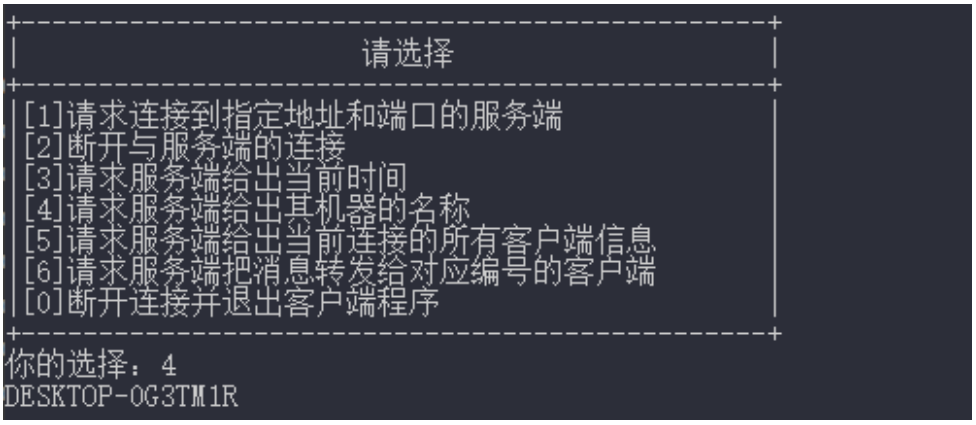
- 客户端选择获取时间功能时，客户端和服务端显示内容截图。

```
+------------------------------------------------+
|                    请选择                      |
+------------------------------------------------+
|[1]请求连接到指定地址和端口的服务端            |
|[2]断开与服务端的连接                          |
|[3]请求服务端给出当前时间                      |
|[4]请求服务端给出其机器的名称                  |
|[5]请求服务端给出当前连接的所有客户端信息      |
|[6]请求服务端把消息转发给对应编号的客户端      |
|[0]断开连接并退出客户端程序                    |
+------------------------------------------------+
你的选择：3
Sat Oct 19 09:55:28 2019
```

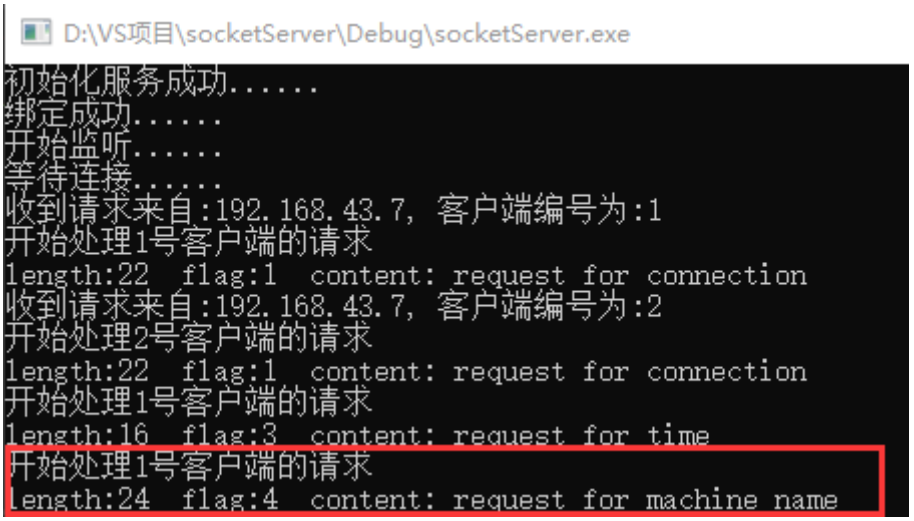
服务端：

- 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：



服务端：

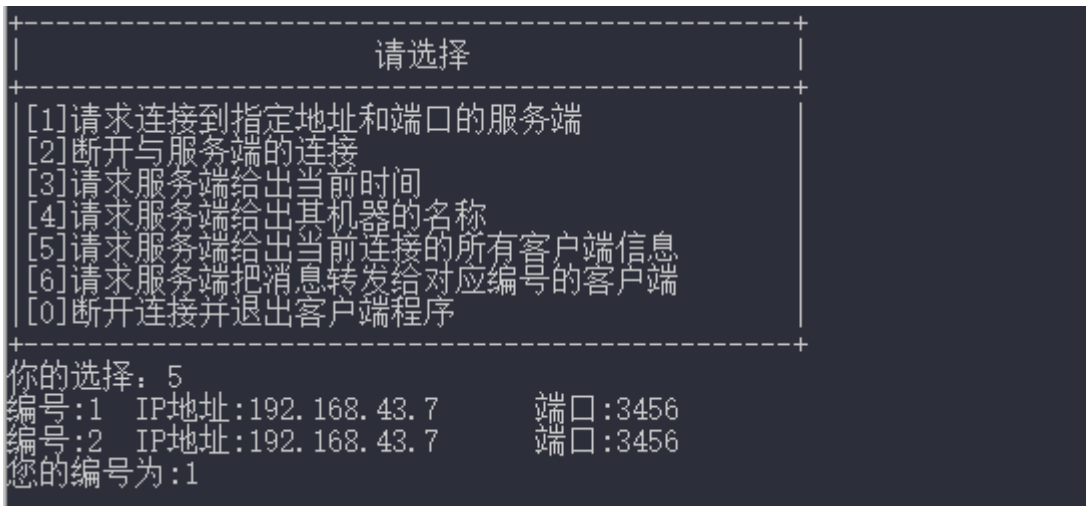


相关的服务器的处理代码片段：

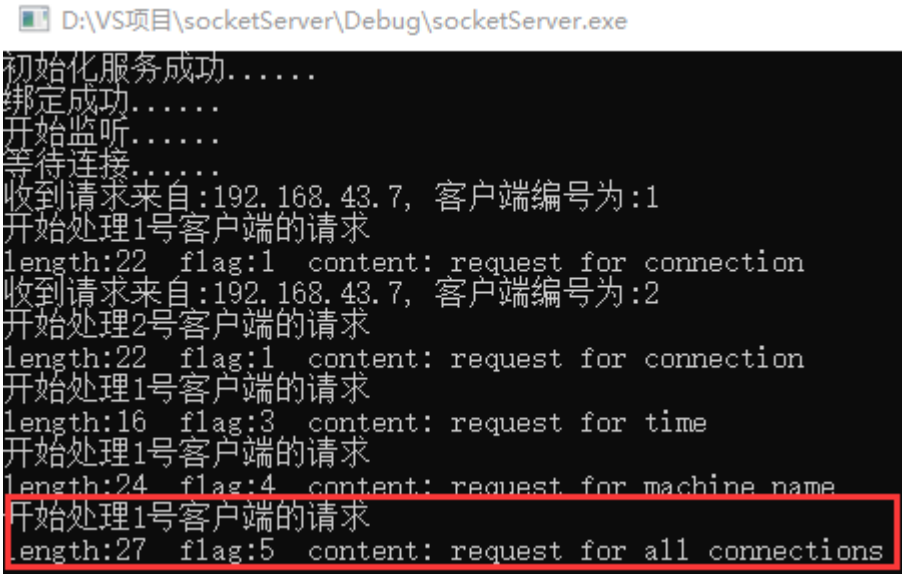
```
void sendServerName(SOCKET client)
{
    CHAR buf[MAX_COMPUTERNAME_LENGTH + 1] = { 0 };
    DWORD bufCharCount = MAX_COMPUTERNAME_LENGTH + 1;
    GetComputerName(buf, &bufCharCount);
    Packet packet;
    setPacket(&packet, 4, buf);
    sendPacket(client, &packet);
}
```

- 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：



  服务端：



  相关的服务器的处理代码片段：

```cpp
void sendConnectedClientList(SOCKET client)
{
    int No = 1;
    int receiverNo;
    string info;
    {
        lock_guard<mutex> lock(mutexLock);
        for (unsigned int i = 0; i < clientList.size(); i++) {
            if (!(clientList[i].getState())) {
                continue;
            }
            if (info.length() + 41 + (to_string(No + 1)).length() > MAXBUFFERSIZE) {
                Packet packet;
                setPacket(&packet, 0, info.c_str());
                sendPacket(client, &packet);
                info.clear();
            }
            if (clientList[i].getClient() == client) receiverNo = No;
            info.append("编号:");
            info.append(to_string(No++));
            info.append(2, ' ');
            info.append("IP地址:");
            info.append(clientList[i].ipAddr);
            info.append(MAXIPLENGTH - strlen(clientList[i].ipAddr), ' ');
            info.append(2, ' ');
            info.append("端口:");
            info.append(to_string(port) + "\n");
        }
    }
    info.append("您的编号为:");
    info.append(to_string(receiverNo) + "\n");
    Packet packet;
    setPacket(&packet, 5, info.c_str());
    sendPacket(client, &packet);
}
```

- 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：

```
+------------------------------------------------+
|                    请选择                      |
+------------------------------------------------+
|[1]请求连接到指定地址和端口的服务端             |
|[2]断开与服务端的连接                           |
|[3]请求服务端给出当前时间                       |
|[4]请求服务端给出其机器的名称                   |
|[5]请求服务端给出当前连接的所有客户端信息       |
|[6]请求服务端把消息转发给对应编号的客户端       |
|[0]断开连接并退出客户端程序                     |
+------------------------------------------------+
你的选择: 6
请输入需要转发的机器id: 2
请输入转发的信息: hello C++!
Successfully send the message
```

服务器端（可选）：

```
D:\VS项目\socketServer\Debug\socketServer.exe
初始化服务成功.....
绑定成功.....
开始监听.....
等待连接.....
收到请求来自:192.168.43.7, 客户端编号为:1
开始处理1号客户端的请求
length:22  flag:1  content: request for connection
收到请求来自:192.168.43.7, 客户端编号为:2
开始处理2号客户端的请求
length:22  flag:1  content: request for connection
开始处理1号客户端的请求
length:10  flag:6  content: hello C++!
```

接收消息的客户端：

```
+-------------------------------------------------+
|                    请选择                       |
+-------------------------------------------------+
|[1]请求连接到指定地址和端口的服务端              |
|[2]断开与服务端的连接                            |
|[3]请求服务端给出当前时间                        |
|[4]请求服务端给出其机器的名称                    |
|[5]请求服务端给出当前连接的所有客户端信息        |
|[6]请求服务端把消息转发给对应编号的客户端        |
|[0]断开连接并退出客户端程序                      |
+-------------------------------------------------+
你的选择：发送者:
编号:1  IP地址:192.168.43.7      端口:3456
hello C++!
```

相关的服务器的处理代码片段：

```
    string info;
    Packet msg;
    {
        lock_guard<mutex> lock(mutexLock);
        for (unsigned int i = 0; i < clientList.size(); i++)
        {
            if (clientList[i].getState()) {
                count++;
            }
            if (clientList[i].getClient() == sender) {
                info.append("发送者:\n");
                info.append("编号:");
                info.append(to_string(count));
                info.append(2, ' ');
                info.append("IP地址:");
                info.append(clientList[i].ipAddr);
                info.append(MAXIPLENGTH - strlen(clientList[i].ipAddr), ' ');
                info.append(2, ' ');
                info.append("端口:3456\n");
            }
        }
        count = 0;
        for (unsigned int i = 0; i < clientList.size(); i++) {

            if (clientList[i].getState()) {
                count++;
                if (count == no) {
                    info.append(packet.content);
                    setPacket(&msg, 1, info.c_str());
                    client = clientList[i].getClient();
                    break;
                }
            }
        }
    }
    if (count < no) return -1;
    char sendData[1024];
    ::memset(sendData, 0, 1024);
    ::memcpy(sendData, &msg, 1024);
    return send(client, sendData, 1024, 0);
```

相关的客户端（发送和接收消息）处理代码片段：

1. 发送消息：

```
case 6:
    cout << "请输入需要转发的机器id: ";
    cin >> id;
    getchar();
    cout << "请输入转发的信息: ";
    cin.getline(msg, MAXBUFFERSIZE);
    sendPacket(msg, 6, id);
    break;
```

2. 接收消息：通过 SocketListener 进行处理

```
void socketListener() {
    while (task) {
        Packet p = receivePacket();
        if (p.flag == 0) {
            cout << "ERROR: 接收失败!\n";
        } else if (p.flag == 2) {
            conn.close();
            task = false;
            cout << "成功断开连接!\n";
        } else {
            cout << p.content << endl;  //normal case
        }
        Sleep(100);
    }
}
```

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？

  答：不需要，因为客户端只需要寻找一个能够使用的端口发送数据包即可，所以其源端口会通过 Socket 的 API 来自动选择一个还没有被占用的端口，每一次调用时客户端的端口不会保持不变。

- 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？

  答：能够连接成功。

- 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？

  答：(1)客户端 IP 地址互不相同时，可以直接通过 IP 地址加以区分。
  (2)IP 地址存在相同的情况时，对不同的 IP 地址使用(1)方法加以区分，对相同的 IP 地址则根据客户端产生的 Socket 描述符加以区分。

- 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）

  答：TCP 的连接状态为 TIME_WAIT，大约持续了 1 分钟左右。

- 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？

  答：服务端的连接状态没有变化，仍然有效。可以尝试向客户端发送一个数据包

观察是否有回应或是设定超时时间，当客户端间隔一定时间没有进行操作时，服务端自动断开连接。

七、 讨论、心得

在本次 Socket 通信实验中，我主要负责的是客户端的代码实现。在开发过程中遇到以下问题：

1. 在实现数据包发送的功能时，在初定义数据包的时候，结构体中各元素都被初始化为 0，原来的代码 send(sclient, req, strlen(req), 0)在执行 strlen (req)时，会受到内存中填充的 0 的影响而导致大小计算错误。因此需要手动传入数据包大小，修改后代码如下：

```
void request(const char* req, const int packetSize) {
    if (cstate != CONNECTED) {
        cout << "当前未连接! " << endl;
        return;
    }
    send(sclient, req, packetSize, 0);
}
```

2. 由于数据接收是由子线程进行，当在主线程中发送了一个数据包时，主线程便接着运行其 while 循环，导致在子线程接受完数据并给出输出结果之前，主线程就已经把下一轮用户的操作列表打印出来，在尝试上锁未果之后，我们选择了使用 Sleep 函数，主线程在每次用户选择完操作后，先休息 1 秒，然后再进行下一轮的操作，以便子线程打印接收的数据。