计算机网络课程实验报告

姓名：汤志文

学号：2111441

专业：物联网工程

实验 1：使用 socket 编写聊天应用程序

一、实验内容：

（1）给出你聊天协议的完整说明。
（2）利用 C 或 C++语言，使用基本的 Socket 函数完成程序。不允许使用 CSocket 等封装后的类编写程序。
（3）使用流式套接字、采用多线程（或多进程）方式完成程序。
（4）程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。
（5）完成的程序应能支持多人聊天，支持英文和中文聊天。
（6）编写的程序应该结构清晰，具有较好的可读性。
（7）在实验中观察是否有数据的丢失，提交源码和实验报告。

二、实验准备：

学习实验所需的相关理论知识：
使用 socket 编程，了解 socket 编程的架构
需要使用多线程编程，了解创建线程的方法
构思非阻塞多线程，使其能够进行多用户聊天

三、实验过程：

（1）聊天协议说明。

用户验证协议：

1. 用户连接到服务器时会标识其 id 号并显示其连接状态。
2. id 号用于标识用户身份。
3. 对用户数量有所限制，当客户端达到上限时输出“客户端数量已满”

用户消息协议：

1. 用户可以向服务器发送消息，消息可以包括中文或英文文本,对文本进行识别，若输入的文本的第一个字符为空格或换行符，则认定为用户输入失误，不对消息进行发送。
2. 服务器上会显示所有用户发送的消息与用户的 id 号，服务器会将其他客户端的 id 及发送的消息转发至客户端。
3. 每条消息包含以下信息：

用户发送的消息内容：用户的文本消息，可以是聊天内容。
用户 id：用于标识消息的发送者，确保其他用户知道消息来源。

用户离线协议：

1. 用户输入“quit”来退出聊天，当用户退出时，服务器端会对变动的用户数量进行显示。

（2）实验设计

服务器端网络连接：

我们采用了常见的套接字编程模式，以便在网络上监听和处理客户端请求。主要涉及下面几个步骤：

1. **初始化网络库**:

WSAStartup 初始化 WinSock 库，显示其连接状态

```
WSAData wsaData;
if (WSAStartup(MAKEWORD(2, 1), &wsaData) == -1) {
    cout << "初始化出错" ;
}
else {
    cout << " WinSock已经初始化" << endl;
}
```

2. **创建套接字**:

使用 socket 函数创建套接字。在这里，我们创建了一个 IPv4 地址族的流式套接字，并传入参数 0 让系统自动选择协议。

```
Server = socket(AF_INET, SOCK_STREAM, 0);
/*
AF_INET：使用ipv4
SOCK_STREAM：套接字类型，确保了数据的完整性和可靠性，流式传输
参数0自动选择协议，这里使用TCP
*/

if (Server == INVALID_SOCKET) // 错误处理
{
    cout<<"socket创建失败";
    return -1;
}
cout << "创建 Socket 成功" << endl;
```

3. **绑定服务器**:

使用 bind 函数将服务器套接字绑定到指定的 IP 地址和端口。

```
// 绑定服务器地址
serverAddr.sin_family = AF_INET;   // 地址类型
serverAddr.sin_port = htons(PORT); // 端口号

if (inet_pton(AF_INET, ipAddr, &(serverAddr.sin_addr)) != 1)//将ServIp转二进制并且存储
{
    cout << "服务端地址绑定出错" << endl;
    return -1;
}
else {
    cout << "服务端addr: "  << ipAddr << " 绑定成功" << endl;
}
if (bind(Server, (LPSOCKADDR)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) // 将服务器套接字与服务器地址和端口绑定
{
    perror("绑定失败");
    exit(EXIT_FAILURE);
}
else
{
    cout << "端口 "  << PORT <<  " 绑定成功" << endl;
}
```

4. **进入监听状态**:

使用 listen 函数将服务器套接字设置为监听状态。

```
// 设置监听/等待队列
if (listen(Server, cMax) != 0)
{
    perror("监听失败");
    return -1;
}
else
{
    cout << "监听成功" << endl;
}

cout << "服务端成功启动" << endl;
```

5. 等待客户端连接：

循环使用 accept 函数接受客户端连接，若 Client 数量没有达到上限，则创建一个新的套接字，用于与客户端通信，若达到上限则输出客户端已满

```
while (true)
{
    if (currentConnect < cMax)
    {
        if (isEmpty() == -1) {
            exit(1);
        }
        int num = isEmpty();
        int addrlen = sizeof(SOCKADDR);
        Clients[num] = accept(Server, (sockaddr*)&clientAddrs[num], &addrlen); // 等待客户端请求

        // 获取客户端ip地址
        char clientIp[INET_ADDRSTRLEN] = "";
        inet_ntop(AF_INET, &(clientAddrs[num].sin_addr), clientIp, INET_ADDRSTRLEN);

        if (Clients[num] == SOCKET_ERROR)
        {
            cout<<"客户端出错";
            closesocket(Server);
            WSACleanup();
            return -1;
        }
        connectCondition[num] = 1;// 连接位置1表示占用
        currentConnect++; // 当前连接数加1

        cout <<  Clients[num] <<  " "  << clientIp <<  " 连接 "    << endl << endl;
        cout << "当前客户端连接数量为: "  << currentConnect <<   endl << endl;

        HANDLE Thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadFunction, (LPVOID)num, 0, NULL); // 创建线程
```

6. 服务器状态输出：

在服务器数量进行变动的时候，输出当前客户端数量

```
else
{
    if (counterr == 1) {
        cout << "客户端数量已满" << endl << endl;
        counterr--;
    }

}
```

客户端网络连接：

客户端是主动发起连接的一方，而服务器处于被动等待连接的状态。使用 connect 函数，客户端尝试连接到服务器。

```
//向服务器发起请求
if (connect(Client, (SOCKADDR*)&servAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
{
    cout << "服务端连接 failed" << endl;
    return -1;
}
else
{
    cout << "服务端连接 success" << endl;
}
```

多线程设计：

在客户端中设置接收消息线程：

```
DWORD WINAPI recvThread() //接收消息线程
{
    while (true)
    {
        char msg[msgSize] = {};//接收消息缓冲区
        if (recv(Client, msg, sizeof(msg), 0) > 0)//参数：客户端套接字，要发送的缓冲区（信息），上一个参数的长度，标志
        {
            cout << endl << msg << endl;
        }
        else if (recv(Client, msg, sizeof(msg), 0) < 0)
        {
            cout << "连接状态异常" << endl;
            break;
        }
    }
    Sleep(10);
    return 0;
}
```

在服务器端设置处理通信的线程：

```
DWORD WINAPI ThreadFunction(LPVOID lpParameter) // 线程
{
    int receByt = 0;
    char recvMsg[msgSize]; // 接收缓冲区
    char sendMsg[msgSize]; // 发送缓冲区
    int num = (int)lpParameter; // 当前连接的索引
    // 发送连接成功的提示消息
    snprintf(sendMsg, sizeof(sendMsg), "你的id是:%d", Clients[num]);
    send(Clients[num], sendMsg, strlen(sendMsg), 0);
    // 循环接收信息
    while (true)
    {
        Sleep(10); // 延时10ms
        receByt = recv(Clients[num], recvMsg, sizeof(recvMsg), 0); // 接收信息

        // 获取客户端ip地址
        char clientIp[INET_ADDRSTRLEN] = "";
        inet_ntop(AF_INET, &(clientAddrs[num].sin_addr), clientIp, INET_ADDRSTRLEN);
        if (receByt > 0) // 接收成功
        {
            cout << "Client [" << Clients[num] << " " << clientIp << "] " << " 发送:" << endl << recvMsg << endl << endl;
            sprintf_s(sendMsg, sizeof(sendMsg), "Id%d发送: %s ", Clients[num], recvMsg); // 格式化发送信息

            for (int i = 0; i < cMax; i++) // 将消息同步到所有聊天窗口
            {
                if (connectCondition[i] == 1 && i != num)
                {
                    send(Clients[i], sendMsg, strlen(sendMsg), 0); // 发送信息
                }
            }
        }
```

客户端消息处理：

消息发送：使用 getline 获取用户输入，若第一个字符为空格或输入为回车则不发送消息，否则将消息发送至服务器端。

```
while (true)
{
    cin.getline(msg, sizeof(msg)); // 使用getline，可以识别到输入间的空格
    if (strcmp(msg, "quit") == 0) // 输入quit断开
    {
        cout << "按任意键继续...";
        cin.get(); // 等待用户按下任意键
        break;

    }
    else if (msg[0] == ' ' || strlen(msg) == 0) // 当输入为空时不发送
    {
        continue;
    }
    send(Client, msg, sizeof(msg), 0); // 向服务端发送
}
```

消息接收：

```
DWORD WINAPI recvThread() //接收消息线程
{
    while (true)
    {
        char msg[msgSize] = {};//接收消息缓冲区
        if (recv(Client, msg, sizeof(msg), 0) > 0)//参数：客户端套接字，要发送的缓冲区（信息），上一个参数的长度，标志
        {
            cout << endl << msg << endl;
        }
        else if (recv(Client, msg, sizeof(msg), 0) < 0)
        {
            cout << "连接状态异常" << endl;
            break;
        }
    }
    Sleep(10);
    return 0;
}
```

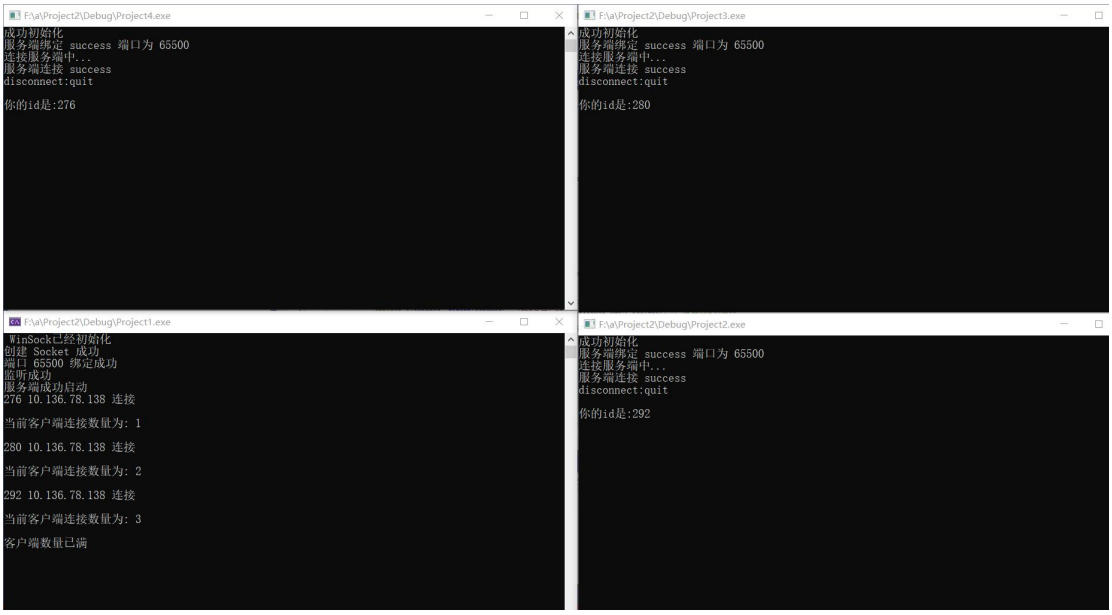
服务器端消息处理：

创建了接收消息线程，循环接受消息，并在服务器端显示的同时，使用 send 函数将数据转发到其他客户端

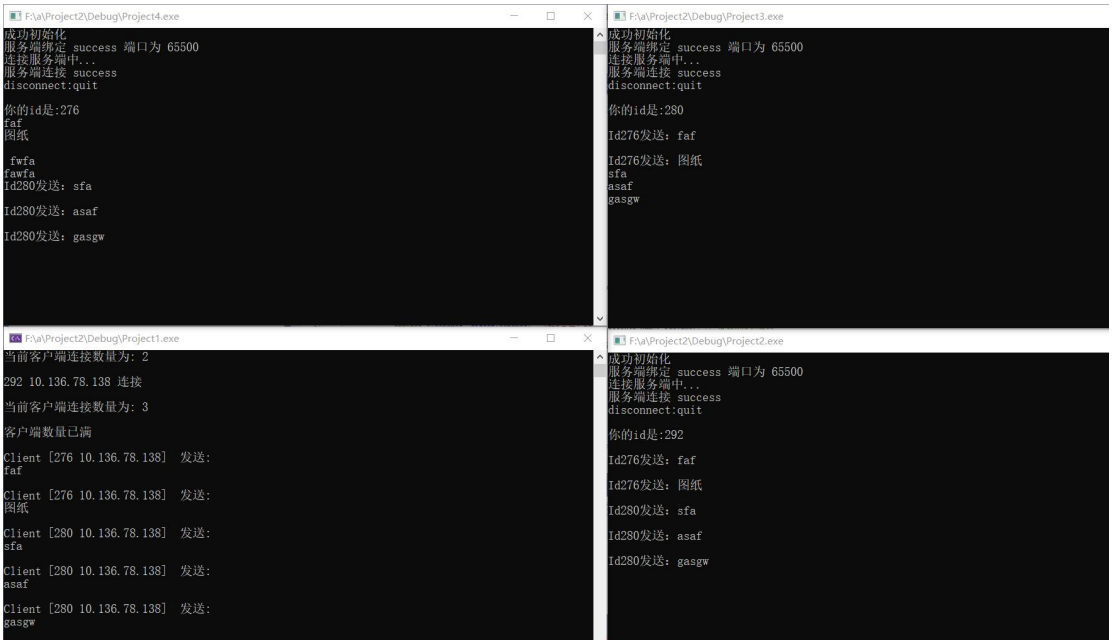
```
DWORD WINAPI ThreadFunction(LPVOID lpParameter) // 线程
{
    int receByt = 0;
    char recvMsg[msgSize]; // 接收缓冲区
    char sendMsg[msgSize]; // 发送缓冲区
    int num = (int)lpParameter; // 当前连接的索引
    // 发送连接成功的提示消息
    snprintf(sendMsg, sizeof(sendMsg), "你的id是:%d", Clients[num]);
    send(Clients[num], sendMsg, strlen(sendMsg), 0);
    // 循环接收信息
    while (true)
    {
        Sleep(10); // 延时10ms
        receByt = recv(Clients[num], recvMsg, sizeof(recvMsg), 0); // 接收信息

        // 获取客户端ip地址
        char clientIp[INET_ADDRSTRLEN] = "";
        inet_ntop(AF_INET, &(clientAddrs[num].sin_addr), clientIp, INET_ADDRSTRLEN);
        if (receByt > 0) // 接收成功
        {
            cout << "Client [" << Clients[num] << " " << clientIp << "] " << " 发送:" << endl << recvMsg << endl << endl;
            sprintf_s(sendMsg, sizeof(sendMsg), "Id%d发送: %s ", Clients[num], recvMsg); // 格式化发送信息

            for (int i = 0; i < cMax; i++) // 将消息同步到所有聊天窗口
            {
                if (connectCondition[i] == 1 && i != num)
                {
                    send(Clients[i], sendMsg, strlen(sendMsg), 0); // 发送信息
                }
            }
        }
```

实验效果演示：

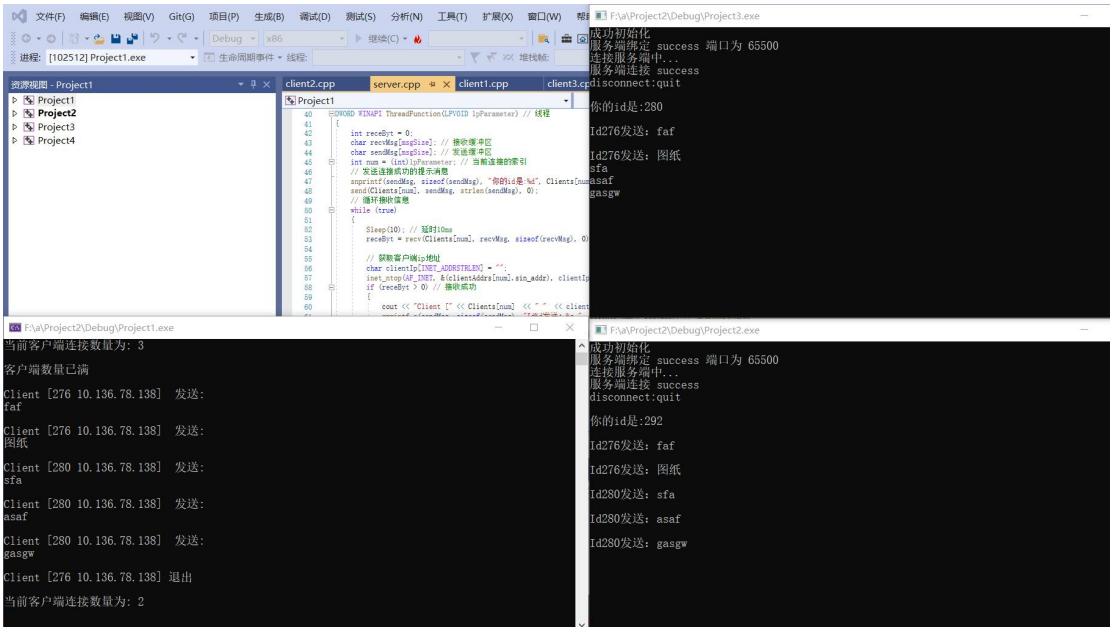
打开三个客户端时，显示客户端已满：



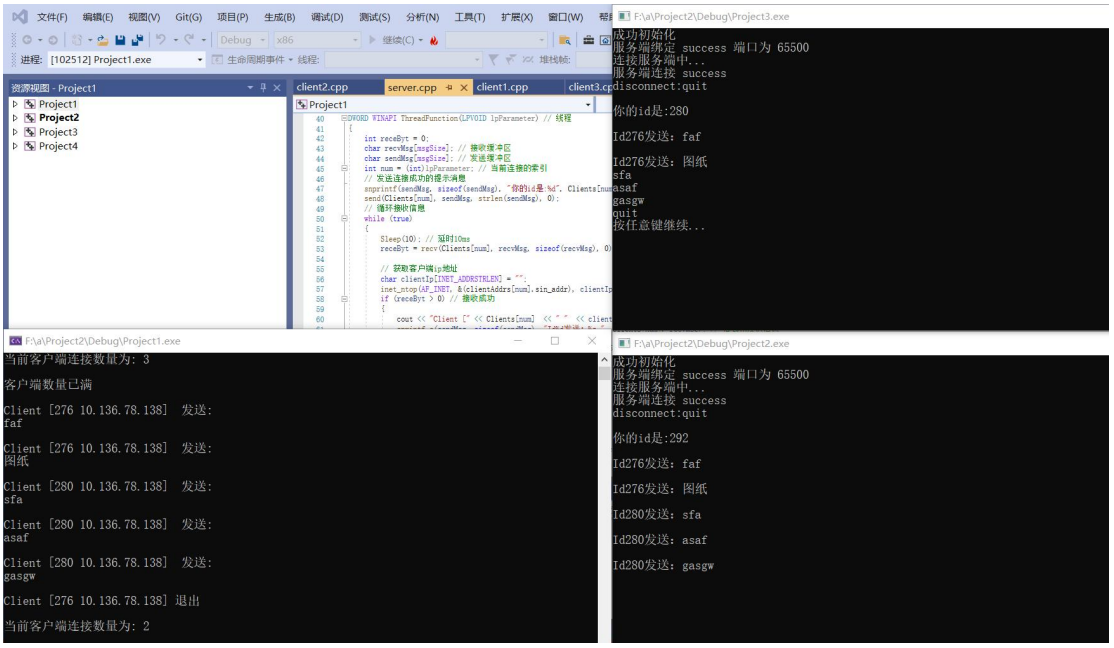
发送消息的演示：



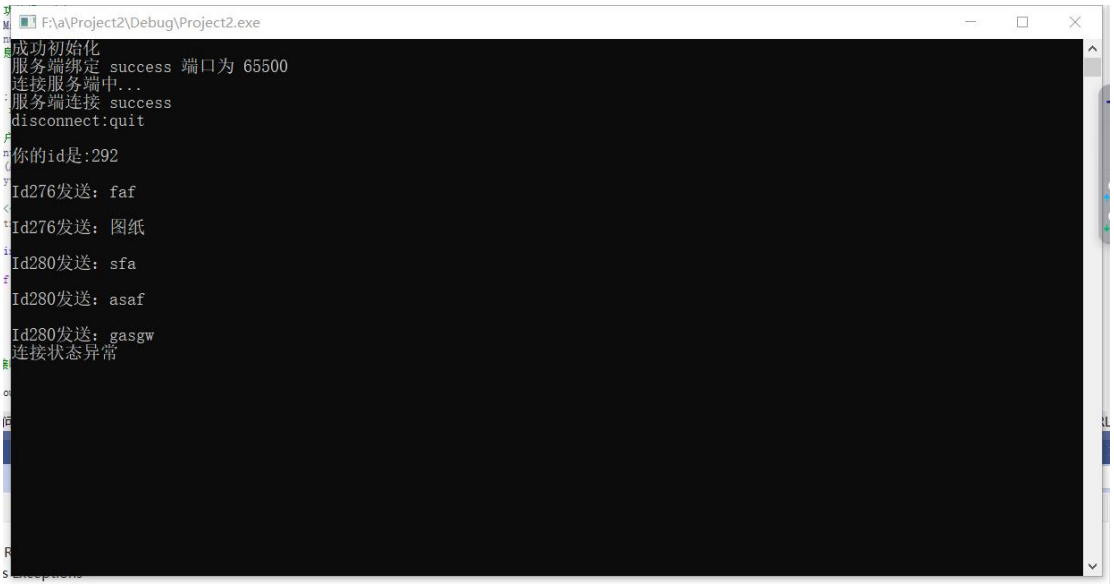
关闭一个端口：



输入 quit 进行退出：



在连接后，将服务器端关闭，客户端报错：



四、实验感悟：

Socket 编程让我更深入地理解了网络通信的基础知识，包括 IP 地址、端口、套接字类型（TCP 和 UDP）等。学习和了解了 c++的 Socket 编程库和 API。这些库包括用于创建、绑定、连接、发送和接收数据的函数。

编写聊天程序需要设计通信协议，包括消息的格式等，这提升了我对通信协议的理解程度。