计算机网络lab1实验报告

计算机学院 2112487 刘轩宇

一 设计思路

（一）基本设计

- 本次实验需要实现多线程通信来实现多人聊天程序，采用TCP协议
- 在实现层面分别建立server端和client端分别实现服务器端和客户端的功能
- 在实现原理上，我按照以下的流程来进行程序设计：

1. 服务器首先运行，等待连接：创建欢迎socket，和本地端口绑定，等待客户端发起连接请求
2. 客户端主动和服务器建立连接：创建客户端本地套接字（隐式捆绑本地端口），指定服务器IP地址和端口号，发起连接请求
3. 服务器接收来自用户端的请求，解除阻塞式等待，返回一个新的socket，与客户端通信
4. 连接API有效，建立起了连接

(二)服务器端

- 在服务器端启动时创建欢迎socket->绑定本地端口->进入监听模式等待客户端连接
- 接受到客户端连接请求后后为其创建线程函数和socket来维护连接，最大接入客户端为MAX_CLIENTS
- 服务器接收来自于客户端的消息并将其广播到其他的客户端

(三)客户端

- 客户端创建socket，请求连接到服务器
- 创建线程函数来接收服务器广播的信息

二 实现代码

服务器端

服务器端主要是模拟实现了一个聊天系统的服务器，主要代码及其解释如下：

```
int main() {
    //初始化socket库
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Error initializing Winsock" << std::endl;
        return 1;
    }

    //服务器端套接字创建
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);//服务器的
socket。AF_INET: AF_INET 表示套接字的地址族，通常用于IPv4网络通信。
    if (serverSocket == INVALID_SOCKET) {
```

```
        cerr << "Error creating server socket" << endl;
        return 1;
    }
    else
    {
        cout << "Create Server Socket successfully!\n" << endl;

    }

    //绑定soket到服务器地址
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;//地址类型为Ipv4
    serverAddr.sin_port = htons(PORT);//绑定服务器端的借口号为12345
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
        cerr << "Error binding server socket" << endl;
        return 1;
    }
    else
    {
        cout << "bind successfully!" << endl;
    }


    if (listen(serverSocket, MAX_CLIENTS) == SOCKET_ERROR) //listen 函数用于将服务
器套接字设置为侦听模式，以接受传入的连接请求。
    {
        cerr << "Error listening on server socket" << endl;
        return 1;
    }

    cout << "Server listening on port " << PORT << endl;



    while (true)
    {

        if (currentnum < MAX_CLIENTS)
        {
            sockaddr_in clientAddr;
            int clientAddrSize = sizeof(clientAddr);
            SOCKET clientSocket = accept(serverSocket, (struct
sockaddr*)&clientAddr, &clientAddrSize);//接受传入的连接请求

            if (clientSocket == INVALID_SOCKET) {
                cerr << "Error accepting client connection" << endl;
                continue;
            }

            int num = find_free();
            isfree[num] = 1;
```

```
            clients[num] = clientSocket;
            currentnum++; //当前连接数加1

            cout << "Client" <<"  " <<num<<"  " << "connected" << endl;

            //thread clientThread(clientHandler, num, clientSocket);
            HANDLE Thread = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)clientHandler, (LPVOID)num, 0, NULL);//创建线程



        }
    }
    closesocket(serverSocket);
    WSACleanup();
    return 0;
}
```

该部分为server端的main函数实现，在while循环之前的部分为服务器端的准备，在服务器端进入监听模式后，进入while循环处理收到的客户端连接请求。当连接成功后，出于客户端管理的需求，我使用find_free函数为每一个接入的客户端赋予编号并进行动态管理。并记录当前连接的用户数。

如下为我们创建的用于接收客户端信息的线程函数

- 使用 recv 函数接收客户端发送的消息。
- 如果接收成功，将消息格式化为特定格式，使用send将信息广播给其他的客户端。

```
DWORD WINAPI clientHandler(LPVOID lpParameter)
{
    int num = (int)lpParameter;
    char sendbuf[1024]; //发送缓冲区
    char buffer[1024];
    while (true)
    {
        memset(buffer, 0, sizeof(buffer));
        int bytesReceived = recv(clients[num], buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            cout << "Client " << num << " disconnected" << endl;
            closesocket(clients[num]);
            isfree[num] = 0; // 标记连接为空闲
            currentnum--;
            break;

        }
        else {
            cout <<"来自于客户"<<num<<"的消息:" << buffer << endl;
            sprintf_s(sendbuf, sizeof(sendbuf), "%s %d: %s ", "来自",num, buffer);
// 格式化发送信息
            // 广播消息给其他所有客户端
            for (int i = 0; i < MAX_CLIENTS; i++) {
                if (i != num && isfree[i] == 1) {
```

```
                    send(clients[i], sendbuf, sizeof(sendbuf), 0);
                }
            }
        }
    }
    return 0;
}
```

客户端

客户端主要实现接收来自服务器端广播的消息并输入在对话框上以及进行消息的输入和发送，main函数代码如下：

```
int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        cerr << "Error initializing Winsock" << std::endl;
        return 1;
    }

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET) {
        cerr << "Error creating client socket" << std::endl;
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    // 使用 inet_pton 将 IP 地址从字符串转换为二进制形式
    if (inet_pton(AF_INET, "127.0.0.1", &(serverAddr.sin_addr)) <= 0) {
        cerr << "Error converting IP address" << std::endl;
        return 1;
    }

    if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr))
== SOCKET_ERROR) {
        cerr << "Error connecting to server" << std::endl;
        return 1;
    }
    else
    {
        cout << "connect to server successfully!" << endl;
    }

    //创建消息线程
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)recvThread, NULL, 0, 0);
    cout << "Enter your message: " << endl;
    while (true)
    {
        string message;
```

```
        getline(cin, message);
        if (strcmp(message.c_str(), "quit") == 0) //输入quit退出
        {
            break;
        }

        if (send(clientSocket, message.c_str(), message.size(), 0) ==
SOCKET_ERROR) {
            cerr << "Error sending message" << std::endl;
            break;
        }
    }

    closesocket(clientSocket);
    WSACleanup();
    return 0;
}
```

在与服务器建立好连接后，首先创建一个消息线程其代码如下：

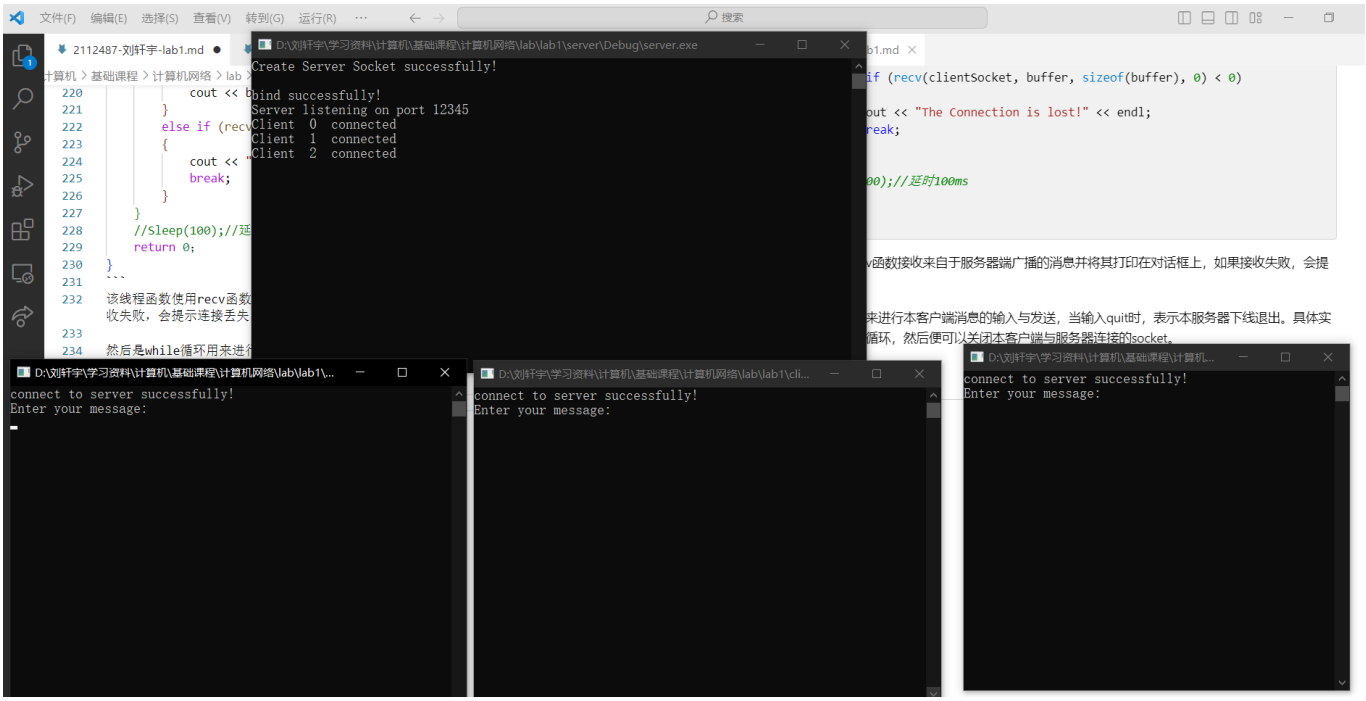
```
DWORD WINAPI recvThread() //接收消息线程
{
    while (true)
    {
        char buffer[BufSize] = {};//接收数据缓冲区
        if (recv(clientSocket, buffer, sizeof(buffer), 0) > 0)
        {
            cout << buffer << endl;
        }
        else if (recv(clientSocket, buffer, sizeof(buffer), 0) < 0)
        {
            cout << "The Connection is lost!" << endl;
            break;
        }
    }
    //Sleep(100);//延时100ms
    return 0;
}
```

该线程函数使用recv函数接收来自于服务器端广播的消息并将其打印在对话框上，如果接收失败，会提示连接丢失。

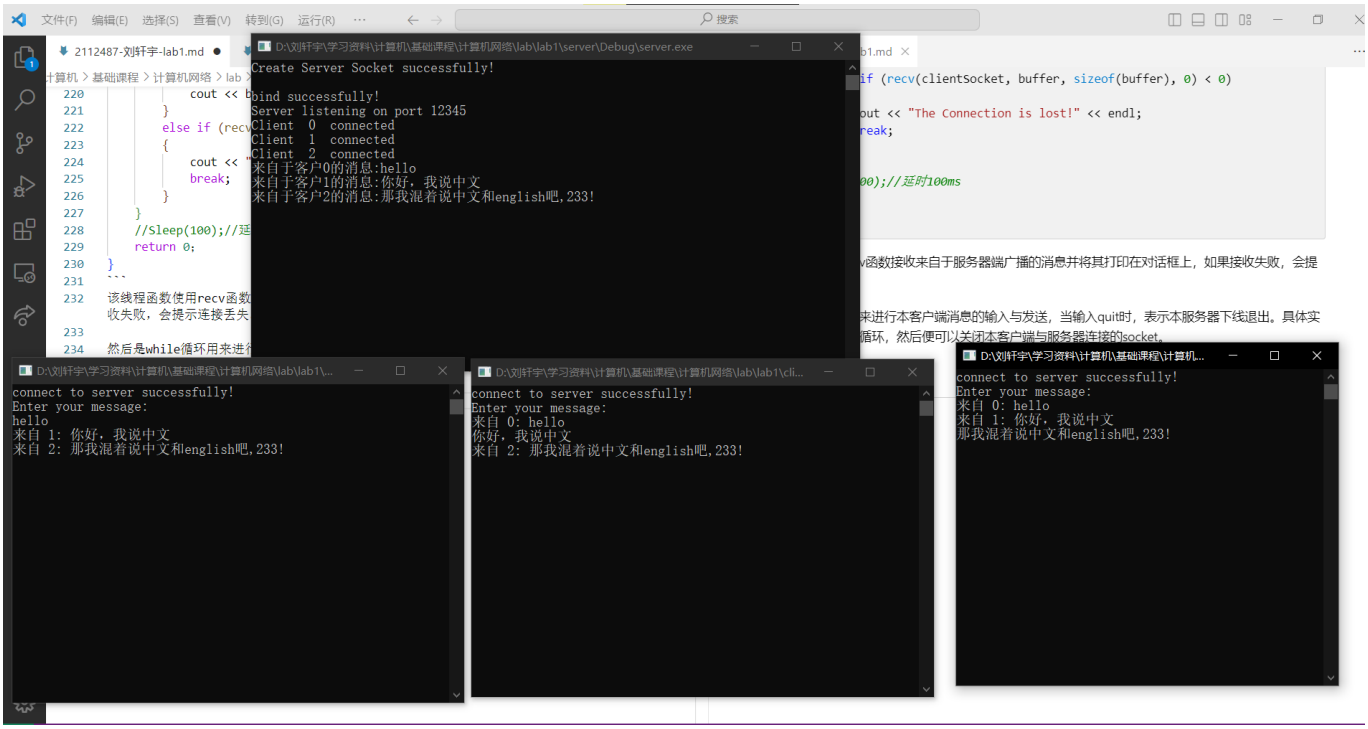
然后是while循环用来进行本客户端消息的输入与发送，当输入quit时，表示本服务器下线退出。具体实现原理为跳出while循环，然后便可以关闭本客户端与服务器连接的socket。

三、运行截图

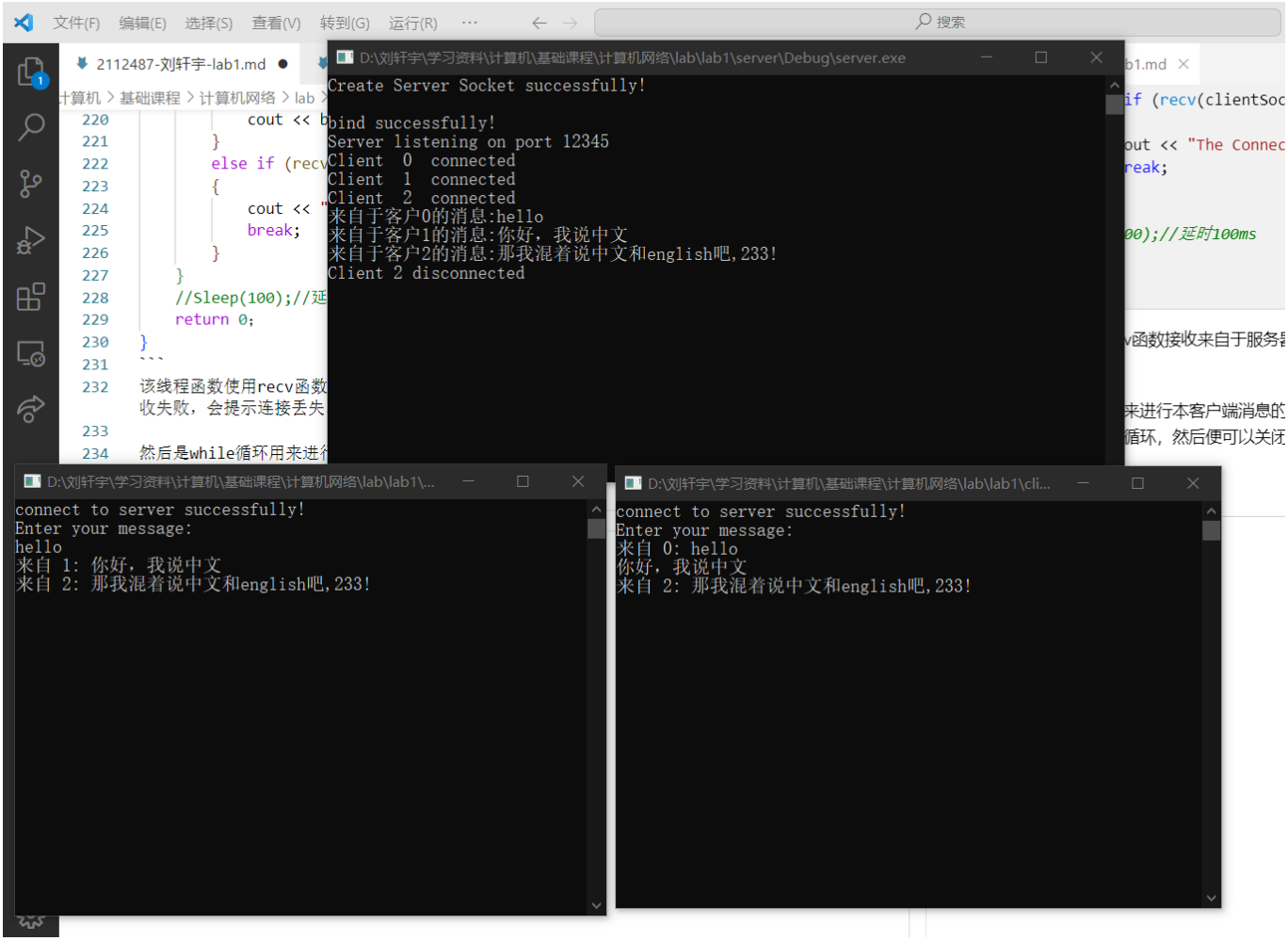
首先是服务器和客户端的启动，可以看到截图如下，服务器正常启动，三个客户端连接。



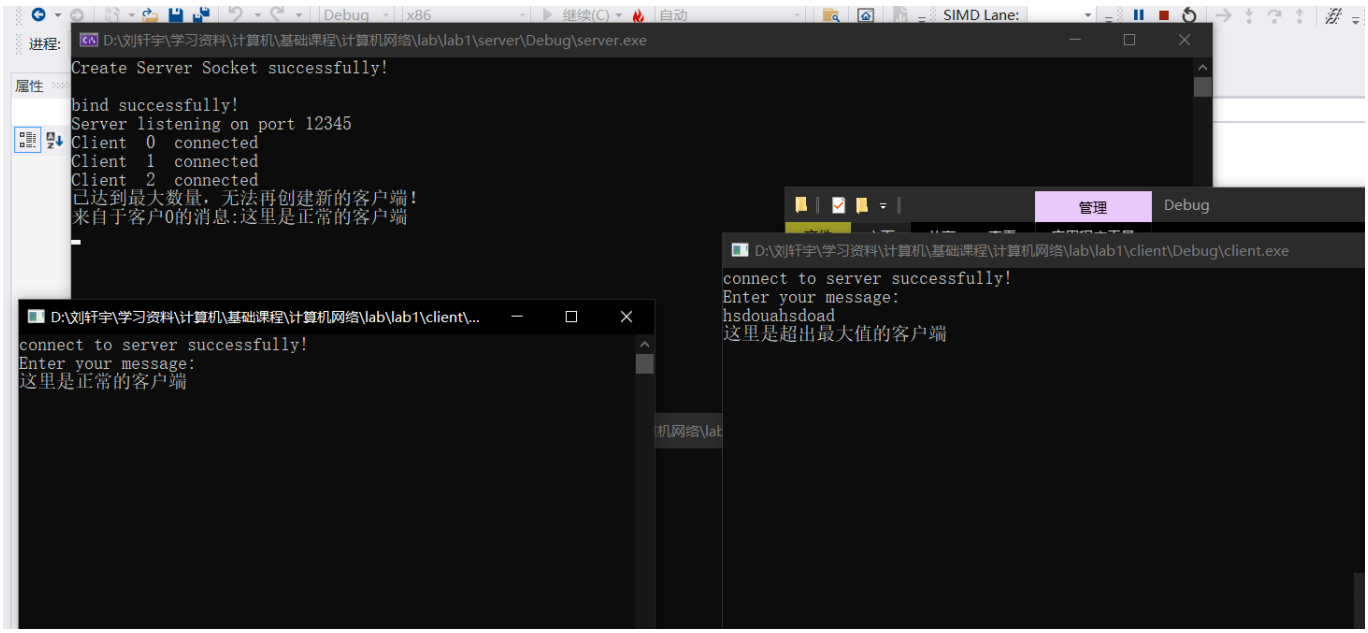
如下展示为正常的信息发送，可以看到发送内容没有丢失，且中英文和数字都可以。



输入quit后，客户端推出。



超出最大连接数后，不能接受的的客户端发送消息无法正常被接受。



github链接：https://github.com/Aurorageek/Computer-Networking/tree/main/lab1