

# 计算机网络实验报告

## Lab1 SOCKET编程

### 网络空间安全学院 信息安全专业

2112492 刘修铭 1063

[https://github.com/lxmliu2002/Computer\\_Networking](https://github.com/lxmliu2002/Computer_Networking)

## 一、协议设计

### (1) 原理

#### 套接字

进程通过套接字发送消息和接受消息。主要包括以下两种类型：

- 数据报套接字：使用UDP协议，支持主机之间面向非连接、不可靠的数据传输
- 流式套接字：使用TCP协议，支持主机之间面向连接的、顺序的、可靠的、全双工字节流传输

#### 协议

计算机网络中各实体之间数据交换必须遵守事先约定好的规则，这些规则称为协议。以下为网络协议的组成要素：

- 语法：数据与控制信息的结构或格式（协议数据单元PDU）
- 语义：需要发出何种控制信息，完成何种动作以及做出何种响应
- 时序：事件实现顺序的详细说明

### (2) 实现

按照实验要求，基于实验目标，采取了以下的协议设计方案：

- 使用TCP传输协议，选用流式套接字，采用多线程方式
- 分别设计两个程序，一个作为服务器端（server），另一个作为客户端（client）。使用时需首先启动 `server.exe`，再启动若干个 `client.exe`，若干个 `client` 借助 `server` 完成通信
- 为了保证通讯质量，本次实验在 `server` 设置了最大连接数 `MaxClient`、最大缓冲区 `BufSize`。当最大连接数上限时，无法再连接；当发送的信息达到消息缓冲区的限制时，超出部分无法发送
- 服务器和客户端之间通过 `send` 和 `recv` 函数来发送和接收消息数据，这些数据的格式和大小都受到语法规则的约束
- 编写了一系列错误处理函数，输出错误信息，方便调试程序

## 服务器端

- 服务器端监听指定端口（在代码中为 8000），等待客户端的连接请求
- 服务器端可以同时接受多个客户端的连接请求，最多支持 `MaxClient - 1` 个客户端同时连接，当达到最大连接数时，不再接受新的连接请求
- 每个客户端连接后，服务器为其创建一个独立的线程（`ThreadFunction` 函数），并使用一个新的 `socket` 来处理该客户端的消息接收和转发
- 服务器的每个线程接收它所负责客户端的消息，并将消息打上时间标签再发送给其他客户端并打印到日志信息
- 服务器维护一个 `clientSockets` 数组，用于存储每个客户端的套接字，以便进行消息的收发

## 客户端

- 客户端创建一个套接字，然后连接到服务器的IP地址和端口号（在代码中为 127.0.0.1 和 8000）
- 客户端也创建一个独立的线程（`recvThread` 函数）来接收服务器发送的消息，并将其显示在控制台上
- 客户端可以通过在控制台中输入文本消息，然后将其发送给服务器

## 消息格式：

- 消息格式为 "`#ClientID`: `Message` --`Timestamp`"，其中 `ClientID` 是客户端的唯一标识，`Message` 是实际消息内容，`Timestamp` 是消息的时间戳
- 客户端发送消息时，服务器接收到消息后会在服务器端控制台显示，并将消息转发给其他所有客户端，以便实现聊天功能
- 服务器端也会接收来自其他客户端的消息，并将其显示在服务器端控制台上

## 退出机制：

- 客户端可以输入 `exit` 来退出聊天程序，此时客户端会关闭连接，并在服务器端显示客户端退出的消息
- 服务器端会检测客户端的连接状态，如果客户端主动关闭连接，服务器会在控制台上显示客户端退出的消息，并关闭相应的套接字，释放资源

# 二、功能实现与代码分析

## (1) 服务器端

该部分实现了一个多线程的聊天服务器，允许多个客户端连接并在客户端之间实现消息广播。通过创建线程处理每个客户端的通信，实现了同时处理多个客户端的连接请求和消息传递。

## 多线程通信

本次实验中，通过编写线程函数，借助宏定义，实现了有连接上限的多线程通信。

代码开头，实现了对连接数、客户端 socket 数组等的定义

```
1  #define PORT 8000 //端口号
2  #define BufSize 1024 //缓冲区大小
3  #define MaxClient 5 //最大连接数 = MaxClient - 1
4
5  SOCKET clientSockets[MaxClient]; //客户端socket数组
6  SOCKET serverSocket; //服务器端socket
7  SOCKADDR_IN clientAddrs[MaxClient]; //客户端地址数组
8  SOCKADDR_IN serverAddr; //定义服务器地址
9
10 int current_connect_count = 0; //当前连接的客户数
11 int condition[MaxClient] = {}; //每一个连接的情况
```

编写了一个线程函数。每个客户端连接都会创建一个线程，函数负责处理该客户端的通信。

- 该函数首先通过传递给线程的参数 `lpParameter` 获取当前连接的 `clientSocket` 索引
- 使用 `recv` 函数接收客户端发送的消息，并根据协议的要求进行处理
- 如果接收成功，将消息格式化为特定格式，并通过 `send` 函数发送给其他连接的客户端，实现消息广播
- 如果客户端主动关闭连接，会通过 `recv` 返回值判断，并在服务器端记录客户端退出的时间

```
1  DWORD WINAPI ThreadFunction(LPVOID lpParameter) //线程函数
2  {
3      int receByt = 0; //接收到的字节数
4      char RecvBuf[BufSize]; //接收缓冲区
5      char SendBuf[BufSize]; //发送缓冲区
6      //char exitBuf[5];
7      //SOCKET sock = *((SOCKET*)lpParameter);
8
9      //循环接收信息
10     while (true)
11     {
12         int num = (int)lpParameter; //当前连接的索引
13         sleep(100); //延时100ms
14         //receByt = recv(sock, RecvBuf, sizeof(RecvBuf), 0);
15         receByt = recv(clientSockets[num], RecvBuf, sizeof(RecvBuf), 0); //接收信
16         息
17         if (receByt > 0) //接收成功
18         {
19             //创建时间戳，记录当前通讯时间
20             auto currentTime = chrono::system_clock::now();
21             time_t timestamp = chrono::system_clock::to_time_t(currentTime);
22             tm localTime;
23             localtime_s(&localTime, &timestamp);
24             char timeStr[50];
```

```

24         strftime(timeStr, sizeof(timeStr), "%Y-%m-%d--%H:%M:%S",
&localTime); // 格式化时间
25         cout << "Client " << clientSockets[num] << " : " << RecvBuf << " --
" << timeStr << endl;
26         sprintf_s(SendBuf, sizeof(SendBuf), "%d: %s --%s ",
clientSockets[num], RecvBuf, timeStr); // 格式化发送信息
27         for (int i = 0; i < MaxClient; i++)//将消息同步到所有聊天窗口
28         {
29             if (condition[i] == 1)
30             {
31                 send(clientSockets[i], SendBuf, sizeof(SendBuf), 0);//发送信
息
32             }
33         }
34     }
35     else //接收失败
36     {
37         if (WSAGetLastError() == 10054)//客户端主动关闭连接
38         {
39             //创建时间戳，记录当前通讯时间
40             auto currentTime = chrono::system_clock::now();
41             time_t timestamp =
chrono::system_clock::to_time_t(currentTime);
42             tm localTime;
43             localtime_s(&localTime, &timestamp);
44             char timeStr[50];
45             strftime(timeStr, sizeof(timeStr), "%Y-%m-%d--%H:%M:%S",
&localTime); // 格式化时间
46             cout << "Client " << clientSockets[num] << " exit! Time: " <<
timeStr << endl;
47             closesocket(clientSockets[num]);
48             current_connect_count--;
49             condition[num] = 0;
50             cout << "current_connect_count: " << current_connect_count <<
endl;
51             return 0;
52         }
53         else
54         {
55             cout << "Failed to receive, Error:" << WSAGetLastError() <<
endl;
56             break;
57         }
58     }
59 }
60 }

```

## main函数

- 在 `main` 函数中，首先进行 `winSock2` 库的初始化，检查初始化是否成功
- 创建服务器套接字 `serverSocket`，并绑定服务器地址和端口
- 设置监听，以便接受客户端连接请求
- 进入循环，不断接受客户端的连接请求，如果连接数未达到最大连接数 `MaxClient`，则创建新的线程来处理客户端通信
- 在 `main` 函数中，也记录了客户端的连接时间和当前连接数，以及处理连接数达到最大值时的情况

```
1  int main()
2  {
3      //system("chcp 936");//防止乱码
4      //初始化DLL
5      WSADATA wsaData;
6      WSASStartup(MAKEWORD(2, 2), &wsaData); //MAKEWORD (主版本号, 副版本号)
7      if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) //错误处
      理 如果初始化成功, wVersion的低位为2, 高位为2, 存储为0x0202
8      {
9          perror("Error in Initializing Socket DLL!\n");
10         exit(EXIT_FAILURE);
11     }
12     cout << "Initializeing Socket DLL is successful!\n" << endl;
13
14     //创建服务器端套接字
15     serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //IPv4地址族, 流式套
      接字, TCP协议
16     if (serverSocket == INVALID_SOCKET) //错误处理
17     {
18         perror("Error in Creating Socket!\n");
19         exit(EXIT_FAILURE);
20     }
21     cout << "Creating Socket is successful!\n" << endl;
22
23     //绑定服务器地址
24     serverAddr.sin_family = AF_INET; //地址类型
25     serverAddr.sin_port = htons(PORT); //端口号
26     if (inet_pton(AF_INET, "127.0.0.1", &(serverAddr.sin_addr)) != 1) {
27         cout << "Error in Inet_pton" << endl;
28         exit(EXIT_FAILURE);
29     }
30     //serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
31     if (bind(serverSocket, (LPSOCKADDR)&serverAddr, sizeof(serverAddr)) ==
      SOCKET_ERROR) //将服务器套接字与服务器地址和端口绑定
32     {
33         perror("Binding is failed!\n");
34         exit(EXIT_FAILURE);
35     }
36     else
37     {
```

```

38     cout << "Binding to port " << PORT << " is successful!\n" << endl;
39 }
40
41 //设置监听
42 if (listen(serverSocket, MaxClient) != 0)
43 {
44     perror("Listening is failed! \n");
45     exit(EXIT_FAILURE);
46 }
47 else
48 {
49     cout << "Listening is successful! \n" << endl;
50 }
51
52 cout << "The server is ready! waiting for client request...\n\n" << endl;
53
54 cout << "current_connect_count: " << current_connect_count << endl;
55
56 //循环接收客户端请求
57 while (true)
58 {
59     if (current_connect_count < MaxClient)
60     {
61         int num = check();
62         int addrlen = sizeof(SOCKADDR);
63         clientSockets[num] = accept(serverSocket,
(sockaddr*)&clientAddrs[num], &addrlen); //接收客户端请求
64         if (clientSockets[num] == SOCKET_ERROR) //错误处理
65         {
66             perror("The client is failed! \n");
67             closesocket(serverSocket);
68             WSACleanup();
69             exit(EXIT_FAILURE);
70         }
71         condition[num] = 1; //连接位 置1表示占用
72         current_connect_count++; //当前连接数加1
73         //创建时间戳，记录当前通讯时间
74         auto currentTime = chrono::system_clock::now();
75         time_t timestamp = chrono::system_clock::to_time_t(currentTime);
76         tm localTime;
77         localtime_s(&localTime, &timestamp);
78         char timeStr[50];
79         strftime(timeStr, sizeof(timeStr), "%Y-%m-%d--%H:%M:%S",
&localTime); // 格式化时间
80         cout << "The client " << clientSockets[num] << " is connected.
Time is " << timeStr << endl;
81         cout << "current_connect_count: " << current_connect_count <<
endl;
82         HANDLE Thread = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)ThreadFunction, (LPVOID)num, 0, NULL); //创建线程

```

```

83         //HANDLE Thread = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)ThreadFunction, (LPVOID) & (clientSockets[num]), 0,
NULL);
84         if (Thread == NULL)//线程创建失败
85         {
86             perror("The Thread is failed!\n");
87             exit(EXIT_FAILURE);
88         }
89         else
90         {
91             CloseHandle(Thread);
92         }
93     }
94     else
95     {
96         cout << "Fulling..." << endl;
97     }
98 }
99
100 closesocket(serverSocket);
101 WSACleanup();
102
103 return 0;
104 }

```

## (2) 客户端

该部分实现了一个简单的客户端程序，用于连接到服务器并进行基本的消息通信。它使用多线程来同时接收和发送消息，允许用户在控制台上输入消息，并将消息发送到服务器。

代码开头部分进行了与服务器端相近的宏定义、全局变量等的声明，此处不再赘述。

### 线程函数

编写了一个接收信息的线程函数 `recvThread`，用于接收从服务器发送过来的消息并显示在控制台上

- 使用 `recv` 函数来接收消息，然后将消息显示在控制台上
- 如果接收到的消息小于等于 0，表示连接已经断开，线程将退出

```

1  DWORD WINAPI recvThread() //接收消息线程
2  {
3      while (true)
4      {
5          char buffer[BufSize] = {}; //接收数据缓冲区
6          if (recv(clientSocket, buffer, sizeof(buffer), 0) > 0)
7          {
8              cout << buffer << endl;
9          }
10         else if (recv(clientSocket, buffer, sizeof(buffer), 0) < 0)
11         {
12             cout << "The connection is lost!" << endl;

```

```

13         break;
14     }
15 }
16 sleep(100); //延时100ms
17 return 0;
18 }

```

## main函数

- 在 `main` 函数中，首先进行 winsock2 库的初始化，检查初始化是否成功
- 创建客户端套接字 `clientSocket`，并连接到服务器
- 如果连接失败，程序会输出错误信息并退出
- 如果连接成功，程序会输出连接成功的信息
- `main` 函数进入一个循环，等待用户输入消息
  - 用户可以通过键盘输入消息，然后使用 `send` 函数将消息发送给服务器
  - 如果用户输入 "exit"，则退出循环，关闭套接字，清理 WinSock2 库并退出程序

```

1  int main()
2  {
3      //system("chcp 936");//防止乱码
4      //初始化DLL
5      WSADATA wsaData;
6      WSStartup(MAKEWORD(2, 2), &wsaData);
7      if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
8      {
9          perror("Error in Initializing Socket DLL!\n");
10         cout << endl;
11         exit(EXIT_FAILURE);
12     }
13     cout << "Initializing Socket DLL is successful!\n" << endl;
14
15     //创建客户端套接字
16     clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
17     if (clientSocket == INVALID_SOCKET)
18     {
19         cout << "Error in Creating Socket!\n" << endl;
20         exit(EXIT_FAILURE);
21         return -1;
22     }
23     cout << "Creating Socket is successful!\n" << endl;
24
25     //绑定服务器地址
26     servAddr.sin_family = AF_INET; //地址类型
27     servAddr.sin_port = htons(PORT); //端口号
28     if (inet_pton(AF_INET, "127.0.0.1", &(servAddr.sin_addr)) != 1) {
29         cout << "Error in Inet_pton" << endl;
30         exit(EXIT_FAILURE);
31     }

```



```

32     //servAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
33
34
35     //向服务器发起请求
36     if (connect(clientSocket, (SOCKADDR*)&servAddr, sizeof(SOCKADDR)) ==
SOCKET_ERROR)
37     {
38         cout << "Error in Connection: " << WSAGetLastError() << endl;
39         exit(EXIT_FAILURE);
40     }
41     else
42     {
43         cout << "Connect is successful! \n" << endl;
44     }
45
46     //创建消息线程
47     CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)recvThread, NULL, 0, 0);
48
49     char buf[BufSize] = {};
50     cout << "Enter a message to send or 'exit' to end the chatting!" << endl;
51
52     //发送消息
53     while (true)
54     {
55         cin.getline(buf, sizeof(buf));
56         if (strcmp(buf, "exit") == 0) //输入exit退出
57         {
58             break;
59         }
60         send(clientSocket, buf, sizeof(buf), 0); //发送消息
61     }
62
63     closesocket(clientSocket);
64     WSACleanup();
65
66
67     return 0;
68 }

```

### (3) 遇到的问题及解决方案

在设计退出程序时，由于先编写的server程序，最开始选择在server端进行信息检测。将收到的消息进行分析，如果是 `exit`，就关闭当前的 `clientSocket[num]`，记录推出时间并发送关闭消息。基于此编写了如下代码：

```

1  for (int i = 0; i < 5; i++)
2  {
3      exitBuf[i] = RecvBuf[i];
4  }
5  if (strcmp(exitBuf, "exit"))
6  {

```

```

7      auto currentTime = chrono::system_clock::now();
8      time_t timestamp = chrono::system_clock::to_time_t(currentTime);
9      tm localTime;
10     localtime_s(&localTime, &timestamp);
11     char timeStr[50];
12     strftime(timeStr, sizeof(timeStr), "%Y-%m-%d--%H:%M:%S", &localTime); // 格式化时间
13     cout << "Client " << clientSockets[num] << " exit! Time: " << timeStr << endl;
14     closesocket(clientSockets[num]);
15     current_connect_count--;
16     condition[num] = 0;
17     send(clientSockets[num], "Your server has been closed!", sizeof(SendBuf), 0);
18     return 0;
19 }

```

但是运行时发现，一旦输入中文，客户端会自动断开连接并退出。修改函数参数类型、控制台编码等均未能改变这一情况。于是我选择将退出机制放在了客户端实现。

```

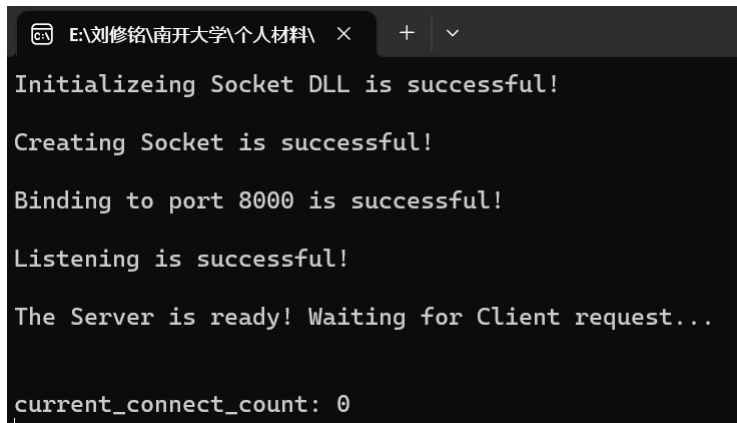
1  if (strcmp(buf, "exit") == 0) //输入exit退出
2  {
3      break;
4  }

```

## 三、运行结果展示

根据实验要求，需实现多人聊天，故此处以三人聊天为示例。

- 首先启动服务器，运行server
  - 控制台已经启动，并输出操作日志
    - Socket DLL初始化成功
    - Socket创建成功
    - 端口号已经绑定
    - 开始监听



```

E:\刘修铭\南开大学\个人材料\ >
Initializeing Socket DLL is successful!
Creating Socket is successful!
Binding to port 8000 is successful!
Listening is successful!
The Server is ready! Waiting for Client request...
current_connect_count: 0

```

- 如果在服务器端未启动的情况下运行客户端，会出现连接失败的错误提示

```
Microsoft Visual Studio 调试 × + ∨

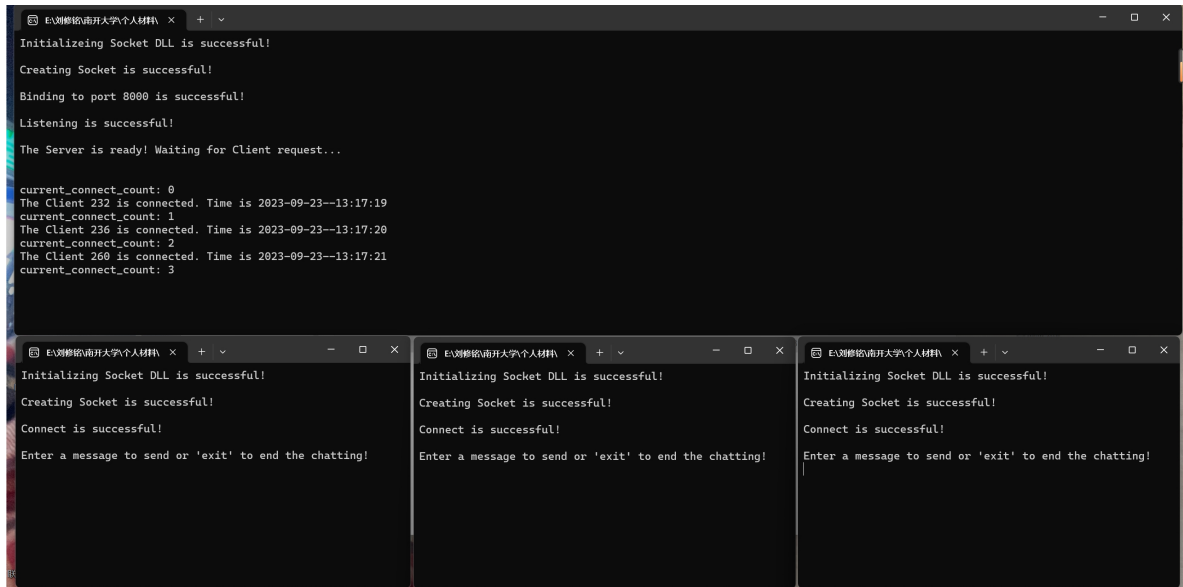
Initializing Socket DLL is successful!

Creating Socket is successful!

Error in Connection: 10061

E:\刘修铭\南开大学\个人材料\课程\2023-2024\debug\client.exe (进程 356)已退出，代码为...
要在调试停止时自动关闭控制台，请启用“工具 > 调试 > 调试时自动关闭控制台”。
按任意键关闭此窗口...|
```

- 接着启动三个客户端程序，模拟三人聊天

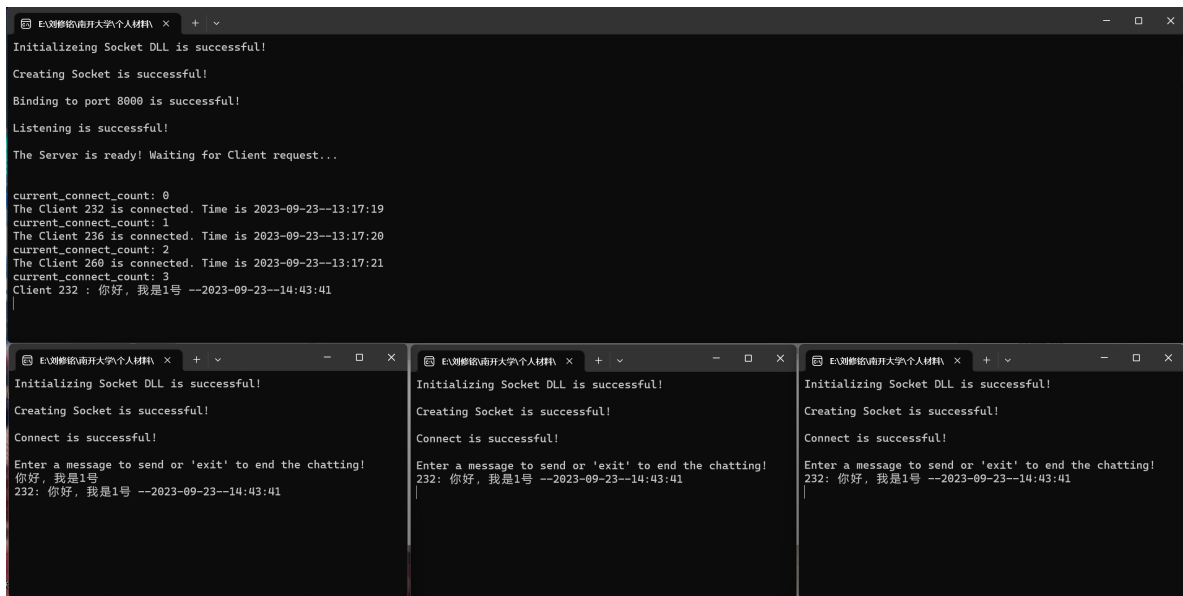


- 客户端分别输出操作日志

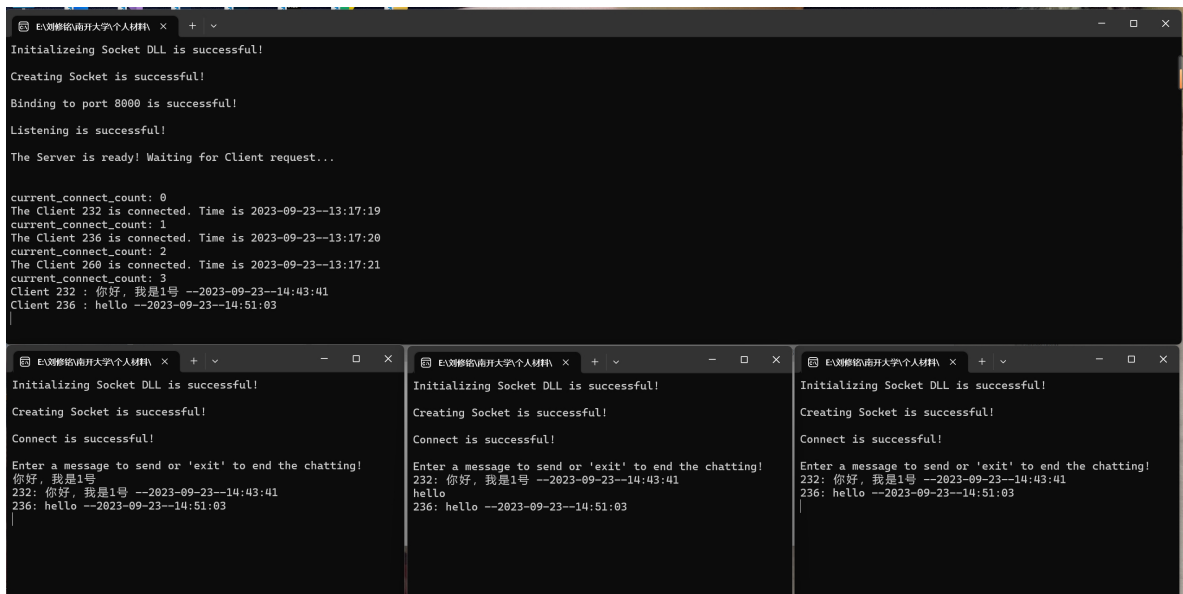
- Socket DLL初始化成功
- Socket创建成功
- 连接成功

- 服务器端显示当前连接数以及连接情况

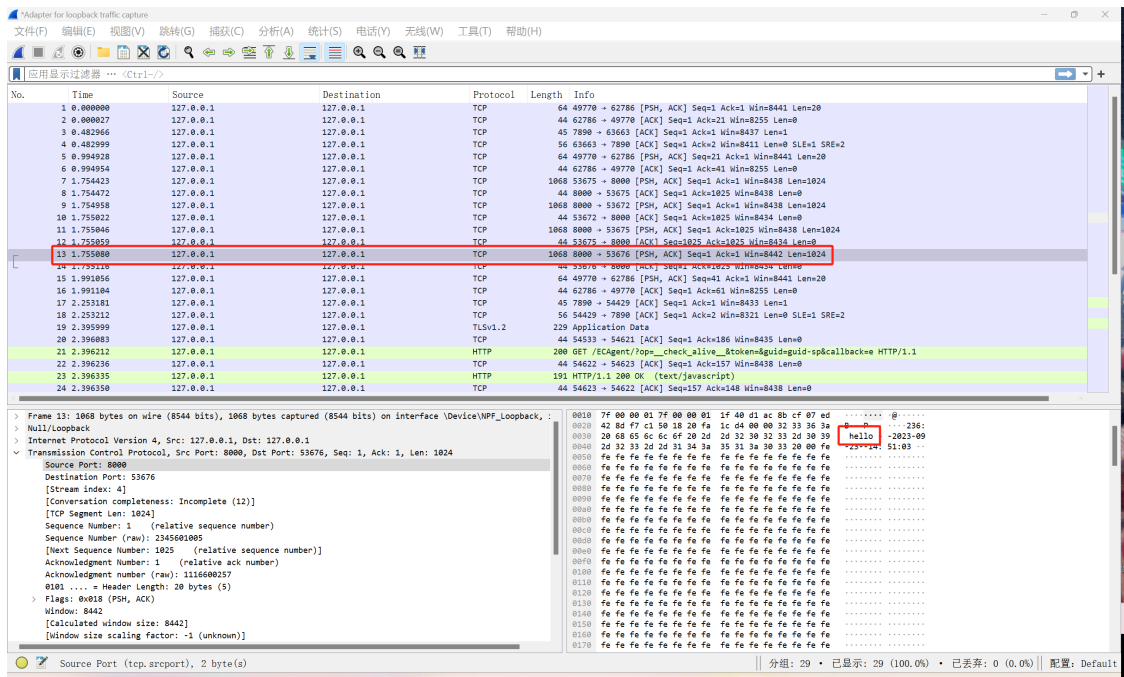
- 在其中一个聊天框中输入 你好，我是1号 并按下回车，可以看到消息发送成功，服务器端输出消息发送内容及时间，所有客户端中同步显示



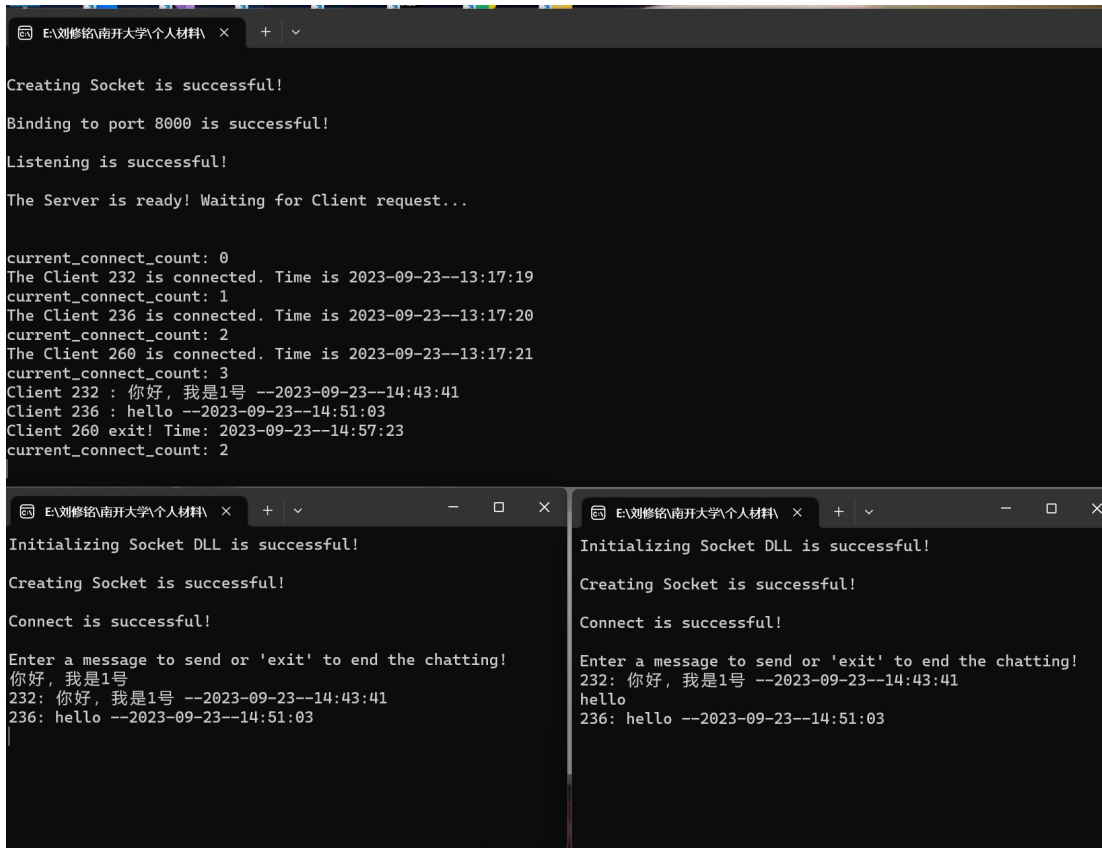
- 在另一个聊天框中输入 `hello` 并按下回车, 可以看到与上面类似的情况



- 借助 `wire shark` 进行数据包抓取, 可以看到如下结果



- 在第三个窗口中输入 exit 退出聊天，可以看到服务端提示该客户端已经退出，并与之断开连接



- 直接关闭第二个聊天窗口，可以看到服务器端也提示该客户端已经退出，并与之断开连接

```
E:\刘修铭\南开大学\个人材料\ x + v

Binding to port 8000 is successful!

Listening is successful!

The Server is ready! Waiting for Client request.

current_connect_count: 0
The Client 232 is connected. Time is 2023-09-23--14:43:41
current_connect_count: 1
The Client 236 is connected. Time is 2023-09-23--14:51:03
current_connect_count: 2
The Client 260 is connected. Time is 2023-09-23--14:57:23
current_connect_count: 3
Client 232 : 你好, 我是1号 --2023-09-23--14:43:41
Client 236 : hello --2023-09-23--14:51:03
Client 260 exit! Time: 2023-09-23--14:57:23
current_connect_count: 2
Client 236 exit! Time: 2023-09-23--14:59:17
current_connect_count: 1

E:\刘修铭\南开大学\个人材料\ x + v -

Initializing Socket DLL is successful!

Creating Socket is successful!

Connect is successful!

Enter a message to send or 'exit' to end the chat:
你好, 我是1号
232: 你好, 我是1号 --2023-09-23--14:43:41
236: hello --2023-09-23--14:51:03
```

- 打开MaxClient个客户端, 可以看到服务器端显示 Fulling..., 表明已经达到最大连接数

```
Microsoft Visual Studio 调试 x + v

The Client 232 is connected. Time is 2023-09-23--13:17:19
current_connect_count: 1
The Client 236 is connected.
current_connect_count: 2
The Client 260 is connected.
current_connect_count: 3
Client 232 : 你好, 我是1号 --2023-09-23--13:17:19
Client 236 : hello --2023-09-23--13:17:19
Client 260 exit! Time: 2023-09-23--13:17:19
current_connect_count: 2
Client 236 exit! Time: 2023-09-23--13:17:19
current_connect_count: 1
The Client 240 is connected.
current_connect_count: 2
The Client 272 is connected.
current_connect_count: 3
The Client 264 is connected.
current_connect_count: 4
The Client 284 is connected.
current_connect_count: 5
Fulling...

E:\刘修铭\南开大学\个人材料\ x + v -

Initializing Socket DLL is successful!

Creating Socket is successful!

Connect is successful!

Enter a message to send or 'exit' to end the chat:
The Connection is lost!

E:\刘修铭\南开大学\个人材料\ x + v -

Initializing Socket DLL is successful!

Creating Socket is successful!

Connect is successful!

Enter a message to send or 'exit' to end the chat:
The Connection is lost!

E:\刘修铭\南开大学\个人材料\ x + v -

Initializing Socket DLL is successful!

Creating Socket is successful!

Connect is successful!

Enter a message to send or 'exit' to end the chat:
The Connection is lost!

E:\刘修铭\南开大学\个人材料\ x + v -

Initializing Socket DLL is successful!

Creating Socket is successful!

Connect is successful!

Enter a message to send or 'exit' to end the chat:
The Connection is lost!
```