

# 实验报告

学号：2112066

姓名：于成俊

## 聊天协议

- 消息类型：
  - 客户端和服务端之间的通信基于文本消息，支持中英文。
- 语法：
  - 每条消息由消息头和消息体组成。消息头包含发送者，即客户端序号，消息体包含文本消息。
- 语义：
  - 发送消息：客户端发送消息，服务器将消息构建为符合协议语法的字符串，然后通过套接字发送给其他客户端。
  - 接收消息：客户端接收消息，将文本消息显示在用户界面上。
  - 日志消息：客户端连接、客户端退出、客户端对话会显示在服务器界面上。
- 时序：
  - 发送消息：客户端发送消息时，将消息放入输出缓冲区，然后通过套接字发送到服务器，然后分发给接收者。
  - 接收消息：接收方客户端会不断监听套接字，来接收来自服务器的消息。一旦接收到消息，会显示在界面上。
- 安全性：
  - 防止多个线程访问资源发生冲突，每个线程访问资源时要上互斥锁。

## 服务器端

### 全局定义

- 设置了两个vector数组，用于存放客户端套接字和客户端序号。
- 设置互斥锁，防止多线程访问客户端列表和客户端序号列表发生冲突。
- 定义服务器监听端口、缓冲区大小
- 设置分配客户端序号的全局计数器

```
// 客户端列表
vector<SOCKET> clients;
//客户端序号列表
vector<int> clientid;
// 用于保护客户端列表的互斥锁
mutex mtx;
// 服务器监听端口
const int PORT = 8080;
//缓冲区大小
const int BUFFER_SIZE = 1024;
// 分配客户端序号的全局计数器
int clientCounter = 0;
```

## Socket编程基本流程

- 1.使用 `WSAStartup` 函数初始化 WinSock，指定使用Winsock 2.2 版本。

```
//初始化WSA(winSock)
WORD sockVersion = MAKEWORD(2, 2); //请求使用 winsock 2.2 版本
WSADATA wsaData; //WSADATA结构体变量的地址值

//成功时会返回0，失败时返回非零的错误代码值
if (WSAStartup(sockVersion, &wsaData) != 0)
{
    cerr << "WSAStartup() error!" << endl;
    return 1;
}
```

- 2.创建服务器套接字，指定使用IPv4、流式套接字、TCP协议

```
//创建套接字
SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //IPv4、
流式套接字、TCP
if (serverSocket == INVALID_SOCKET) //INVALID_SOCKET: 无效套接字的特殊值
{
    cerr << "Failed to create server socket." << endl;
    return 1;
}
```

- 3.绑定套接字

```
//绑定IP和端口
sockaddr_in serverAddr; //IPv4的指定方法是使用struct sockaddr_in类型的变量
serverAddr.sin_family = AF_INET; //IPv4
serverAddr.sin_addr.S_un.S_addr = INADDR_ANY; //IP地址设置成INADDR_ANY，让系
统自动获取本机的IP地址
serverAddr.sin_port = htons(PORT); //设置端口。htons将主机字节序转换为网络字节顺
序
//bind函数把一个地址族中的特定地址赋给socket。
if (bind(serverSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR)
{
    cerr << "Bind failed." << endl;
    return 1;
}
```

- 4.使用 `listen()` 函数进行监听：`SOMAXCONN` 是一个宏，表示在等待连接队列中允许的最大连接数。这个宏通常被定义为系统默认的最大值。（助教所提问题）

```
//开始监听
if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR)
{
    cout << "listen error !" << endl;
    return 1;
}

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

- 5.进入死循环，不断接收客户端连接。连接成功后，创建新线程来处理客户端消息，防止发生阻塞。

其中，`detach()` 的作用是将新线程设置为分离状态，这意味着当线程结束时，它的资源会被自动释放，无需等待父线程的结束。（助教所提问题）

```
while (true) {
    // 接受客户端连接
    SOCKET client = accept(serverSocket, NULL, NULL);
    if (client == INVALID_SOCKET) {
        cerr << "Accept failed." << endl;
    }
    else {
        // 给客户端分配序号
        AssignClientIndex(client);
        // 创建一个新线程来处理客户端消息
        thread clientThread(ReceiveMessages, client);
        clientThread.detach();
    }
}
```

## 主要函数

- 为客户端分配序号，并将序号发给相应的客户端，并将新接受的客户端加入vector中，为了防止发生冲突，在此作用域上了互斥锁 `lock_guard<mutex> lock(mtx)`

```
// 给客户端分配序号
void AssignClientIndex(SOCKET client) {

    // 为客户端分配唯一的序号
    int clientIndex = clientCounter++;
    lock_guard<mutex> lock(mtx);
    clients.push_back(client);
    clientid.push_back(clientIndex);
    // 向客户端发送分配的序号
    cout << "Client " << clientIndex << " connected " << endl;
    string message = "You are assigned index " + to_string(clientIndex);
    send(client, message.c_str(), message.size(), 0);

}
```

- 服务器接收来自客户端的消息，首先在vector中确定该客户端的序号，从而将消息和客户端序号一并发给除发消息的客户端之外的其他客户端。

其中，查找客户端使用了vector中的 `find()` 函数，复杂度为  $O(n)$ ，这会导致当连接的客户端数量过多时，会产生阻塞现象。改进方案：① 让客户端发消息的时候，直接将它的序号发过来，然后服务器对消息进行处理，从而获取它的客户端号。② 利用线程池，由于每个客户端都占有自己的线程，发消息之后，通过线程池的管理，直接获取相关的信息。（助教所提的问题、自己的思考）

```
// 服务器接收消息的线程
void ReceiveMessages(SOCKET client) {
    char buffer[BUFFER_SIZE];
    // 寻找客户端的索引
    int clientIndex;
    {
        lock_guard<mutex> lock(mtx);
```

```

        auto it = find(clients.begin(), clients.end(), client);
        if (it != clients.end()) {
            // 找到了套接字, it 是指向找到元素的迭代器
            size_t index = distance(clients.begin(), it);
            clientIndex = clientid[index];
        }
    }
    while (true) {
        int bytesReceived = recv(client, buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            // 客户端断开连接
            lock_guard<mutex> lock(mtx);
            clients.erase(remove(clients.begin(), clients.end(), client),
clients.end());
            clientid.erase(remove(clientid.begin(), clientid.end(),
clientIndex), clientid.end());
            closesocket(client);
            cout << "Client " << clientIndex << " 断开连接" << endl;
            break;
        }
        // 构建包含客户端序号的消息
        string message = "Client " + to_string(clientIndex) + ": " +
string(buffer, bytesReceived);
        lock_guard<mutex> lock(mtx);
        cout << message << endl;
        // 广播消息给所有客户端 (除发送消息的客户端)
        for (SOCKET otherClient : clients) {
            if (otherClient != client) {
                send(otherClient, message.c_str(), message.size(), 0);
            }
        }
    }
}

```

## 客户端

### 全局定义

- 定义服务器监听端口、缓冲区大小

```

// 服务器监听端口
const int PORT = 8080;
//缓冲区大小
const int BUFFER_SIZE = 1024;

```

### Socket编程基本流程

- 初始化WinSock、创建套接字 (与服务器端一样)
- 配置服务器地址信息

```

sockaddr_in serverAddr;           //服务器地址信息
serverAddr.sin_family = AF_INET;  //使用IPv4地址信息
// 使用 inet_pton 将 IP地址转化为二进制形式
const char* serverIP = "127.0.0.1";
if (inet_pton(AF_INET, serverIP, &serverAddr.sin_addr) <= 0) {
    cerr << "Invalid server IP address." << endl;
    return 1;
}
serverAddr.sin_port = htons(PORT); //将端口号从主机字节序（小端字节序）转换为网络
字节序（大端字节序）

```

- 连接服务器，若成功，输出“Connected to the server.”

```

// 连接到服务器
if (connect(clientSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR) {
    cerr << "Failed to connect to the server." << endl;
    return 1;
}

cout << "Connected to the server." << endl;

```

- 启动消息接收线程，并用死循环来实现随时发送消息，输出“quit”可以退出，断开与服务器的连接。

```

// 启动消息接收线程
thread messageThread(receiveAndDisplayMessages, clientSocket);
messageThread.detach();

while (true) {
    string message;
    cout << "Enter your message (or type 'quit' to exit):\n";
    getline(cin, message);

    if (message == "quit") {
        break;
    }

    send(clientSocket, message.c_str(), message.size(), 0);
}

```

## 主要函数

- 用于接收和显示来自服务器的消息

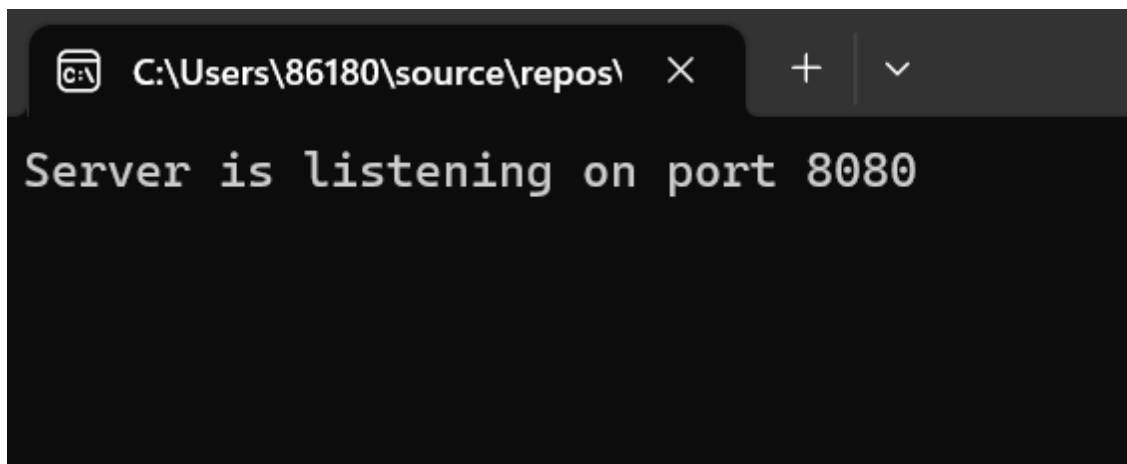
```

void receiveAndDisplayMessages(SOCKET clientSocket) {
    char buffer[BUFFER_SIZE];
    while (true) {
        int bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (bytesReceived > 0) {
            buffer[bytesReceived] = '\0'; // 在接收的数据后添加 null 终止字符
            cout << buffer << endl;
        }
    }
}

```

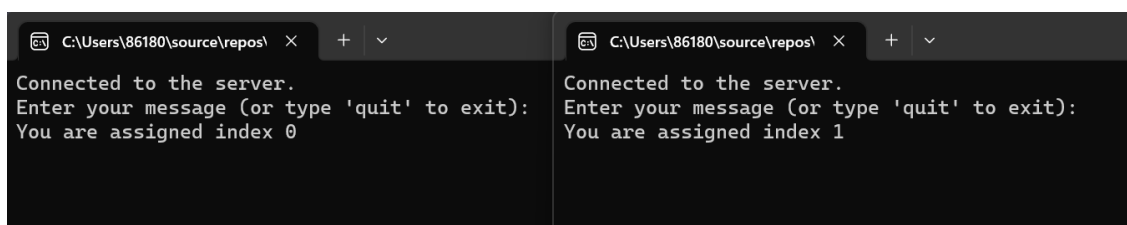
## 运行演示

- 首先启动服务器，即运行“服务器.exe”
  - 服务器端会显示它在哪个端口监听



```
C:\Users\86180\source\repos\ > Server is listening on port 8080
```

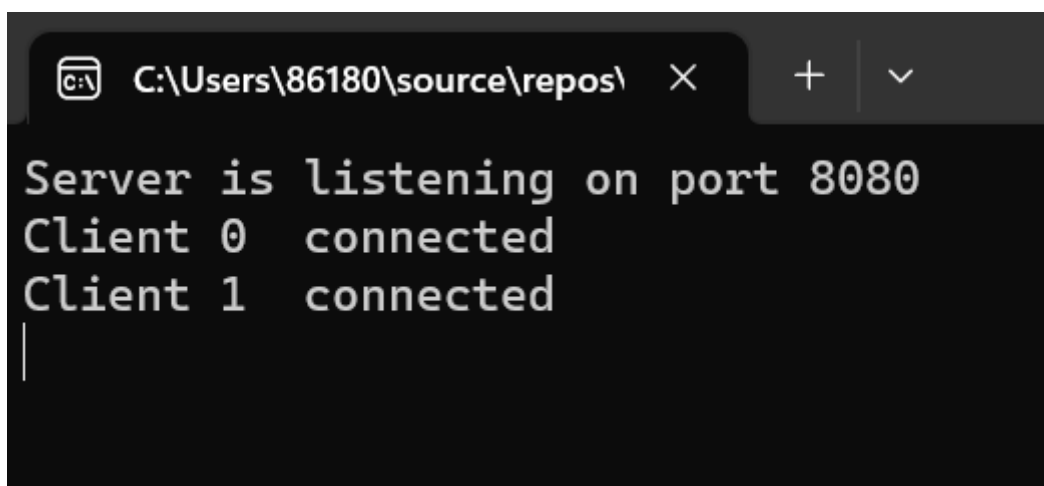
- 然后启动两个客户端，即同时运行两个“客户端.exe”
  - 若连接成功，则显示Connected to the server
  - 提示你回车发送消息，并且输入quit可以退出
  - 显示你的客户端号



```
C:\Users\86180\source\repos\ > Connected to the server.  
Enter your message (or type 'quit' to exit):  
You are assigned index 0
```

```
C:\Users\86180\source\repos\ > Connected to the server.  
Enter your message (or type 'quit' to exit):  
You are assigned index 1
```

- 服务器端会显示哪个客户端连接上了



```
C:\Users\86180\source\repos\ > Server is listening on port 8080  
Client 0 connected  
Client 1 connected  
|
```

- 哪个客户端都可以发送消息，客户端会先发送给服务器，然后服务器将消息广播给除发送消息之外的其他客户端。客户端收到的消息都会被标识是哪个客户端发过来的消息。服务器这边会显示所有的消息记录。

```
C:\Users\86180\source\repos\ X + v
Connected to the server.
Enter your message (or type 'quit' to exit):
You are assigned index 0
Client 1: 你好
hello
Enter your message (or type 'quit' to exit):
我是 client 0
Enter your message (or type 'quit' to exit):
Client 1: 我是client 1

C:\Users\86180\source\repos\ X + v
Connected to the server.
Enter your message (or type 'quit' to exit):
You are assigned index 1
你好
Enter your message (or type 'quit' to exit):
Client 0: hello
Client 0: 我是 client 0
我是client 1
Enter your message (or type 'quit' to exit):

C:\Users\86180\source\repos\ X + v
Server is listening on port 8080
Client 0 connected
Client 1 connected
Client 1: 你好
Client 0: hello
Client 0: 我是 client 0
Client 1: 我是client 1
```

- 客户端输入“quit”，则退出，断开连接，服务器端会显示哪个客户端断开连接了。

```
C:\Users\86180\source\repos\ X + v
Connected to the server.
Enter your message (or type 'quit' to exit):
You are assigned index 0
Client 1: 你好
hello
Enter your message (or type 'quit' to exit):
我是 client 0
Enter your message (or type 'quit' to exit):
Client 1: 我是client 1
quit
请按任意键继续. . .

C:\Users\86180\source\repos\ X + v
Connected to the server.
Enter your message (or type 'quit' to exit):
You are assigned index 1
你好
Enter your message (or type 'quit' to exit):
Client 0: hello
Client 0: 我是 client 0
我是client 1
Enter your message (or type 'quit' to exit):
quit
请按任意键继续. . .

C:\Users\86180\source\repos\ X + v
Server is listening on port 8080
Client 0 connected
Client 1 connected
Client 1: 你好
Client 0: hello
Client 0: 我是 client 0
Client 1: 我是client 1
Client 1 断开连接
Client 0 断开连接
```

## 实验过程遇到的问题及分析

分散在各段代码讲解中。