

南 开 大 学

计算机网络课程实验报告

实验一：Socket 聊天程序



专 业_____信息安全_____

学 号_____2113662_____

姓 名_____张丛_____

班 级 _____信息安全一班_____

一、实验要求

- 1.给出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。不得使用 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.程序应该有基本的对话界面，但可以不是图形界面。程序应该有正常的退出方式。
- 5.完成的程序应该支持多人聊天，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丢失，提交可执行文件、程序源码和实验报告。

二、实验原理

1.套接字：

套接字用于发送或接收数据，实现网络通信。

流套接字：面向连接的套接字。提供了可靠的、基于连接的、面向流的数据传输，通常使用传输控制协议（TCP）。

数据报套接字：也称为面向消息的套接字。这种套接字提供了无连接的、不可靠的数据传输，通常使用用户数据报协议（UDP）

本次实验涉及的对套接字的操作：

- 1.创建套接字：通过 `socket()` 系统调用创建套接字。
- 2.绑定套接字：通过 `bind()` 系统调用将套接字与一个特定的本

地地址和端口绑定。

3.监听连接请求（仅适用于服务器）：对于服务器应用，使用 ``listen()`` 函数。

4.接受连接（仅适用于服务器）：使用 ``accept()`` 函数。

5.连接到远程主机（仅适用于客户端）：使用 ``connect()`` 函数。

6.发送和接收数据：使用 ``send()`` 和 ``recv()`` 函数。

7.关闭套接字：使用 ``closesocket()`` 函数。

2.协议：

一组规则和约定，用于控制数据通信和交换。协议定义了数据的格式、传输方法、错误检测和纠正，以及通信过程中各个实体的行为。

三、实验过程

1.协议设计

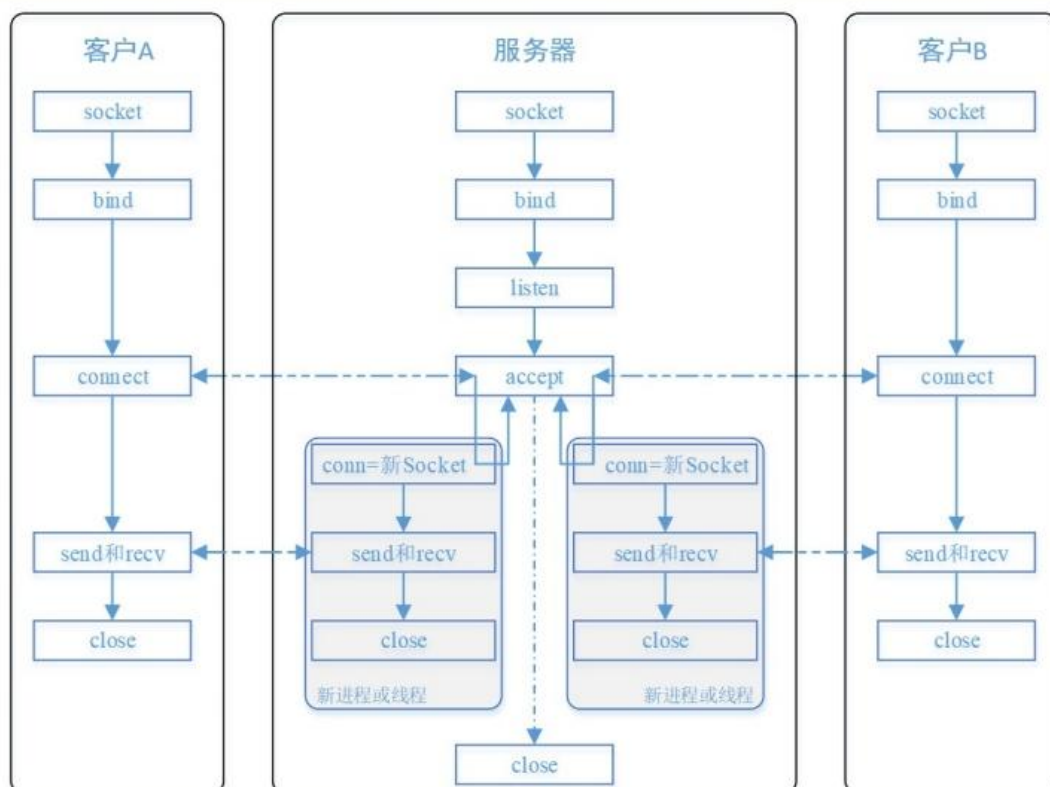
- 使用 TCP 传输协议，选用流式套接字，采用多线程方式；
- 存在服务端和客户端，客户端借助服务端进行通信；
- 打印的消息格式为消息来源-消息-时间；
- 指定端口为本地地址的 6000；
- 服务端存在最大连接数，当连接数量达到上限时，无法再连接；
- 消息存在最大消息缓冲区，超出限制时，超出部分无法发送；

- 服务器端会为每个客户端分别创建线程；
- 服务端将打印所有信息，生成日志，并附有时间；
- 服务端将检测客户端的状态；
- 当连接过程出现错误时，输出错误信息；
- 服务端和客户端均可通过 exit 退出

2. 程序流程

程序的通信流程可参考理论课所学：

2.3 利用TCP服务的应用程序编写步骤



3. 程序实现

服务器端（Server）

● recvThread 线程

主要功能：

- ◆ 在服务端接受客户端请求后建立的线程；
- ◆ 通过参数获取客户端索引，使用 recv 函数接收此客户端消息；
- ◆ 打印日志；
- ◆ 通过 send 函数转发消息到其他客户端；
- ◆ 处理客户端断开连接的情况。

代码实现：

```
1.  DWORD WINAPI  recvThread(LPVOID lpParameter)
2.  {
3.      int get_rec = 0;
4.      char RecvBuf[BufSize]; //接收缓冲区
5.      char SendBuf[BufSize]; //发送缓冲区
6.
7.      //接收信息
8.      while (true)
9.      {
10.         int num = (int)lpParameter;    //当前连接的索引，在调用线程函数时传入
11.         Sleep(100);
12.
13.         get_rec = recv(clientSockets[num], RecvBuf, sizeof(RecvBuf), 0); //接收信息
14.         if (get_rec > 0)                //接收成功
15.         {
16.             char ctime[50];              //获取时间
```

```

17.         memset(ctime, 0, sizeof(ctime));
18.         get_current_time(ctime);
19.
20.         //打印消息在服务器窗口
21.         cout << "Client " << num << ": " << RecvBuf << " " << ctime << endl;
22.         // 格式化要转发的信息
23.         sprintf_s(SendBuf, sizeof(SendBuf), "Client %d: %s %s", num, RecvBuf, ctime);
24.
25.         for (int i = 0; (i < MaxClient); i++)           //将消息转发到所有聊天窗口（但不包括自己）
26.         {
27.             if ((is_online[i] == 1)&&i!=num)           //在线
28.             {
29.                 send(clientSockets[i], SendBuf, sizeof(SendBuf), 0);
30.             }
31.         }
32.     }
33.     else //接收失败
34.     {
35.         char ctime[50];
36.         memset(ctime, 0, sizeof(ctime));
37.         get_current_time(ctime);
38.         cout << "——Client " << num << " 离线啦... " << ctime << endl;
39.
40.         closesocket(clientSockets[num]);           //释放此套接字
41.         total_connect--;                             //在线总数减一
42.         is_online[num] = 0;
43.         cout << "——在线人数: " << total_connect << endl;
44.         return 0;
45.     }
46. }
47. }

```

● sendThread 线程

主要功能：

- ◆ 向客户端发送服务端本身的消息；

◆ exit 退出机制。

代码实现：

```
1.  DWORD WINAPI sendThread(LPVOID lpParameter)
2.  {
3.      char msg[BufSize] = {};
4.      char SendBuf[BufSize] = {};
5.      while (1)
6.      {
7.          cin.getline(msg, sizeof(msg));
8.          if (strcmp(msg, "exit") == 0) // 输入 exit 退出
9.          {
10.             break;
11.          }
12.
13.          char ctime[50];
14.          memset(ctime, 0, sizeof(ctime));
15.          get_current_time(ctime);
16.
17.          sprintf_s(SendBuf, sizeof(SendBuf), "Server: %s %s", msg, ctime); // 格式化服务器消息
18.          for (int i = 0; i < MaxClient; i++)
19.          {
20.              if (is_online[i])
21.              {
22.                  send(clientSockets[i], SendBuf, sizeof(SendBuf), 0); // 发送消息
23.              }
24.          }
25.      }
26.      return 0;
27. }
```

● main 主函数

主要功能：

- ◆ 套接字初始化，声明版本；
- ◆ 创建服务端套接字；
- ◆ 绑定服务器地址；
- ◆ 设置监听；
- ◆ 接收客户端请求，创建相应线程；
- ◆ 资源的释放。

代码实现：

```
1.  int main()
2.  {
3.      // 套接字初始化
4.      WSADATA wsaData;
5.      //WSADATA 被用来存储被WSAStartup 函数调用后返回的 Windows Sockets 数据。
6.      WORD sockVersion = MAKEWORD(2, 2);
7.      // 声明采用 2.2 版本
8.      if (WSAStartup(sockVersion, &wsaData) != 0)
9.      {
10.         cout << "一套接字初始化失败!" << endl;
11.         return 0;
12.     }
13.
14.     // 创建服务器端套接字
15.     SOCKET serverSocket;
16.     serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //IPv4 地址族，流式套接字，
        TCP 协议
17.     if (serverSocket == INVALID_SOCKET)
18.     {
19.         cout << "创建服务器套接字失败" << endl;
20.         return 0;
21.     }
22.     cout << "已创建服务器端套接字" << endl;
23.
24.     // 绑定服务器地址
25.     SOCKADDR_IN serverAddr;           // 服务器地址
```



```

26.     serverAddr.sin_family = AF_INET;          //地址类型为IPv4
27.     serverAddr.sin_port = htons(PORT);        //端口号
28.     serverAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY); //本机 IP 地址 , INADDR_ANY 表示服
    务器将接受来自任何源地址的连接请求
29.     //htonl 将主机字节序转换为网络字节序
30.
31.     if (bind(serverSocket, (LPSOCKADDR)&serverAddr, sizeof(serverAddr)) == SOCKET_ERRO
R) //将服务器套接字与服务器地址和端口绑定
32.     {
33.         cout << "绑定失败" << endl;
34.         return 0;
35.     }
36.     else
37.     {
38.         cout << "已绑定服务器端套接字 " << endl;
39.     }
40.
41.     //设置监听
42.     if (listen(serverSocket, SOMAXCONN) != 0)
43.     {
44.         cout << "监听失败" << endl;
45.         return 0;
46.     }
47.     else
48.     {
49.         cout << "正在监听..." << endl;
50.     }
51.
52.     //接收客户端请求
53.     while (1)
54.     {
55.         if (total_connect >= MaxClient)
56.         {
57.             cout << "-- 已达到最大连接数..." << endl;
58.             continue;
59.         }
60.
61.         int num;          //找到可以放入的数组索引
62.         for (int i = 0; i < MaxClient; i++)
63.         {
64.             if (is_online[i] == 0)

```

```

65.         {
66.             num = i;
67.             break;
68.         }
69.     }
70.     int address_len = sizeof(SOCKADDR);           //接收客户端请求
71.     SOCKET revClientSocket = accept(serverSocket, (sockaddr*)&clientAddrs[num], &a
        ddress_len);
72.     if (revClientSocket == SOCKET_ERROR)
73.     {
74.         cout << "—客户端连接失败" << endl;
75.         return 0;
76.     }
77.     else
78.     {
79.         clientSockets[num] = revClientSocket;    //客户端Socket 加入数组
80.         is_online[num] = 1;                      //在线转态
81.         total_connect++;                          //当前连接总数加1
82.     }
83.
84.     char ctime[50];
85.     memset(ctime, 0, sizeof(ctime));
86.     get_current_time(ctime);
87.
88.     cout << "—Client " << num << " 上线啦!    " << ctime << endl;
89.     cout << "—目前在线人数: " << total_connect << endl;
90.
91.     HANDLE recv_Thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)recvThread,
        (LPVOID)num, 0, NULL); //接收、转发线程
92.     if (recv_Thread == NULL) //线程创建失败
93.     {
94.         cout << "—创建 recv 线程失败" << endl;
95.     }
96.     HANDLE send_Thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)sendThread,
        NULL, 0, NULL); //server 发送消息线程
97.     if (send_Thread == NULL)
98.     {
99.         cout << "—创建 send 线程失败" << endl;
100.    }
101.
102.    }

```

```

103.
104.     closesocket(serverSocket);
105.     WSACleanup();
106.     cout << "—已结束服务器" << endl;
107.
108.     return 0;
109. }

```

客户端（Client）

● recvThread 线程

主要功能：

- ◆ 在客户端连接到服务端后创建的线程；
- ◆ 通过 recv 函数接收服务端消息；

代码实现：

```

1.  DWORD WINAPI recvThread() //接收消息线程
2.  {
3.      while (1)
4.      {
5.          char buffer[BufSize] = {}; //接收数据缓冲区
6.          if (recv(clientSocket, buffer, sizeof(buffer), 0) > 0) //接收
7.          {
8.              cout << buffer << endl;
9.          }
10.         else if (recv(clientSocket, buffer, sizeof(buffer), 0) < 0)
11.         {
12.             cout << "—连接失败..." << endl;
13.             break;
14.         }
15.     }
16.     Sleep(100);
17.     return 0;

```

18. }

● main 主函数

主要功能：

初始化，创建客户端套接字，绑定地址；

向服务端发起请求；

创建 recvThread 线程；

通过 send 函数发送消息；

exit 退出机制；

代码实现：

```
1.  int main()
2.  {
3.      WSADATA wsaData;
4.      WSStartup(MAKEWORD(2, 2), &wsaData);
5.
6.      clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
7.
8.      //绑定服务器地址
9.      SOCKADDR_IN servAddr;    //服务器地址
10.     servAddr.sin_family = AF_INET;    //地址类型
11.     servAddr.sin_port = htons(PORT); //端口号
12.     if (inet_pton(AF_INET, "127.0.0.1", &(servAddr.sin_addr)) != 1) //将字符串 "127.0.0.1" 转换为二进制形式的 IPv4 地址并存储在 servAddr.sin_addr 中
13.     {
14.         cout << "—无法绑定地址" << endl;
15.         exit(EXIT_FAILURE);
16.     }
17.
18.     //向服务器发起请求
```

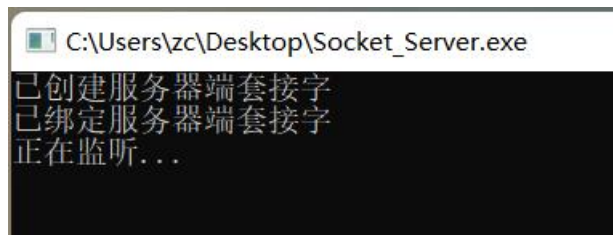
```

19.     if (connect(clientSocket, (SOCKADDR*)&servAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
20.     {
21.         cout << "—无法连接到服务器: " << WSAGetLastError() << endl;
22.         exit(EXIT_FAILURE);
23.     }
24.
25.     // 创建消息线程
26.     CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)recvThread, NULL, 0, 0);
27.
28.
29.     char msg[BufSize] = {};
30.     cout << "—欢迎加入聊天! " << endl;
31.
32.     // 发送消息
33.     while (1)
34.     {
35.         cin.getline(msg, sizeof(msg));
36.         if (strcmp(msg, "exit") == 0) // 输入 exit 退出
37.         {
38.             break;
39.         }
40.         send(clientSocket, msg, sizeof(msg), 0); // 发送消息
41.     }
42.
43.     closesocket(clientSocket);
44.     WSACleanup();
45.
46.     return 0;
47. }

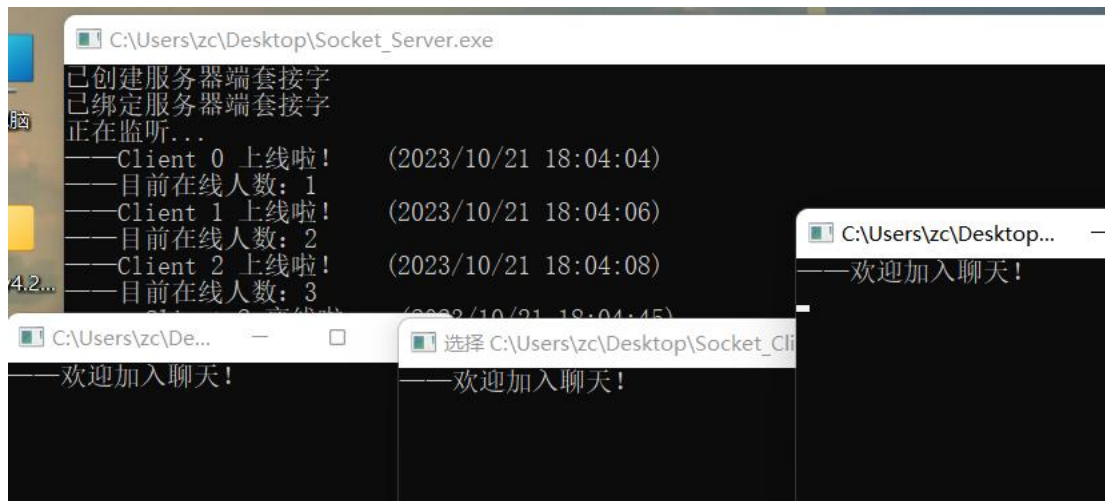
```

4. 程序测试

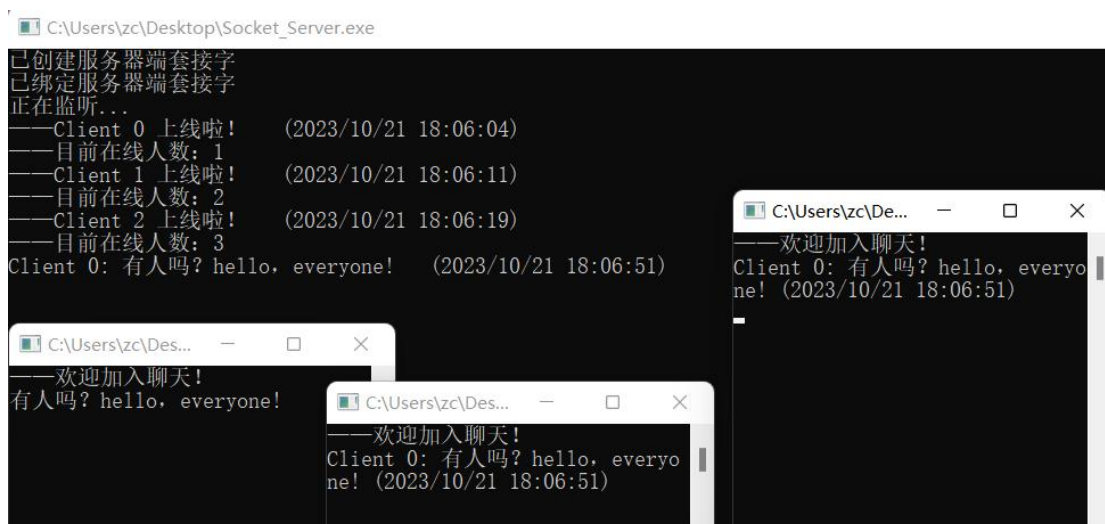
启动服务端：

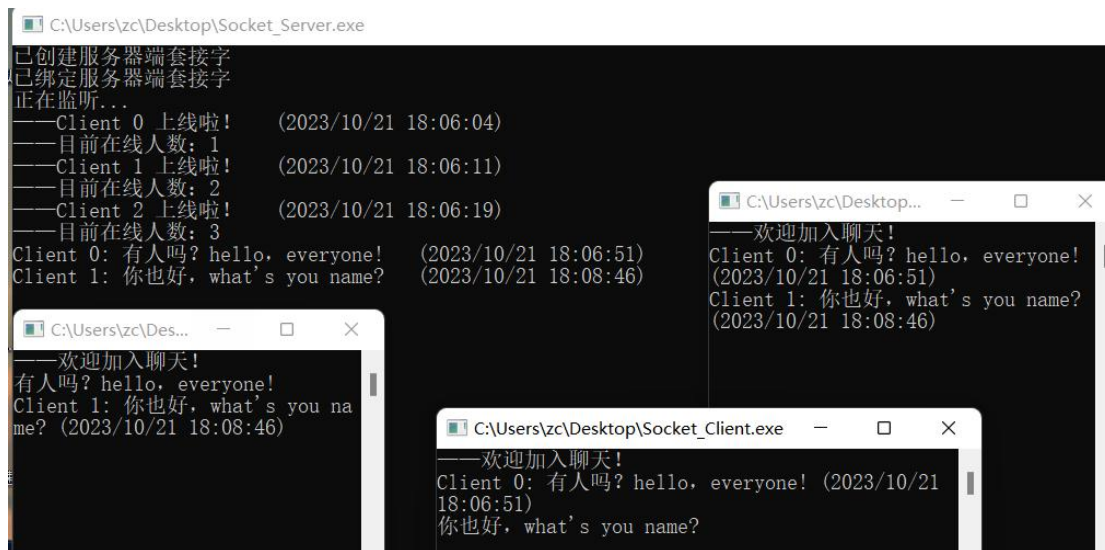


然后启动三个客户端：



客户端发送消息：





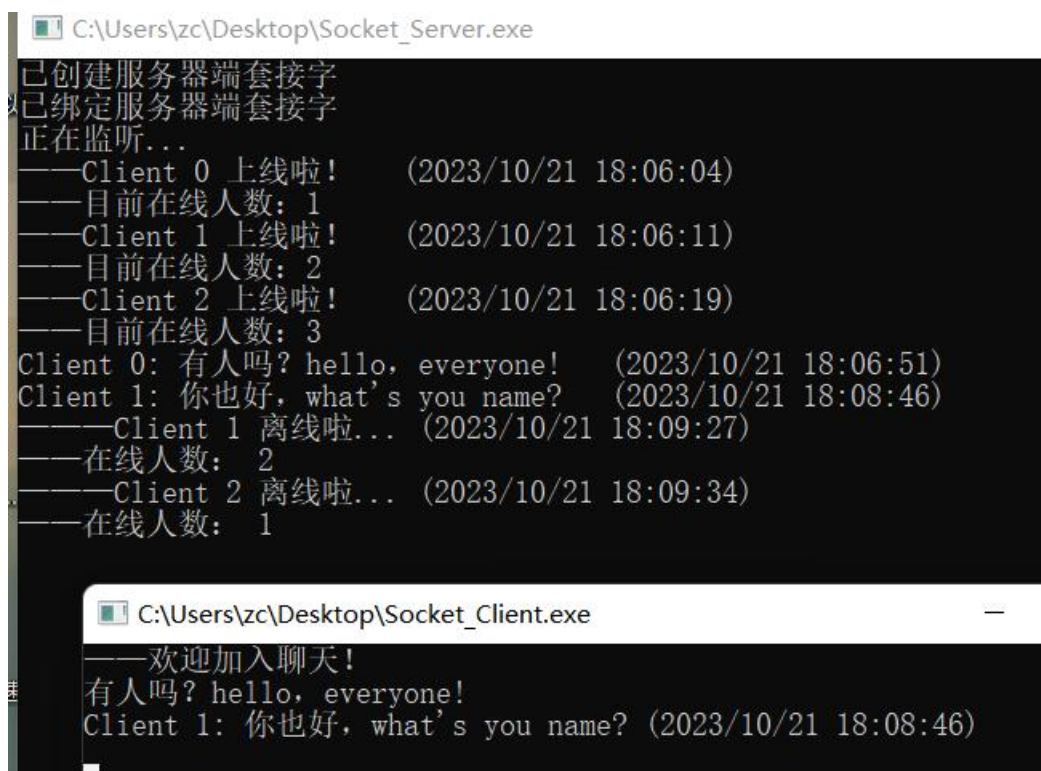
```
C:\Users\zc\Desktop\Socket_Server.exe
已创建服务器端套接字
已绑定服务器端套接字
正在监听...
——Client 0 上线啦! (2023/10/21 18:06:04)
——目前在线人数: 1
——Client 1 上线啦! (2023/10/21 18:06:11)
——目前在线人数: 2
——Client 2 上线啦! (2023/10/21 18:06:19)
——目前在线人数: 3
Client 0: 有人吗? hello, everyone! (2023/10/21 18:06:51)
Client 1: 你也好, what's you name? (2023/10/21 18:08:46)

C:\Users\zc\Desktop\Socket_Client.exe
——欢迎加入聊天!
有人吗? hello, everyone! (2023/10/21 18:06:51)
Client 1: 你也好, what's you name? (2023/10/21 18:08:46)

C:\Users\zc\Desktop\Socket_Client.exe
——欢迎加入聊天!
Client 0: 有人吗? hello, everyone! (2023/10/21 18:06:51)
你也好, what's you name?
```

可见, 服务端打印日志, 其他客户端接收到消息并附来源和时间信息。

客户端断开连接 (exit) :



```
C:\Users\zc\Desktop\Socket_Server.exe
已创建服务器端套接字
已绑定服务器端套接字
正在监听...
——Client 0 上线啦! (2023/10/21 18:06:04)
——目前在线人数: 1
——Client 1 上线啦! (2023/10/21 18:06:11)
——目前在线人数: 2
——Client 2 上线啦! (2023/10/21 18:06:19)
——目前在线人数: 3
Client 0: 有人吗? hello, everyone! (2023/10/21 18:06:51)
Client 1: 你也好, what's you name? (2023/10/21 18:08:46)
——Client 1 离线啦... (2023/10/21 18:09:27)
——在线人数: 2
——Client 2 离线啦... (2023/10/21 18:09:34)
——在线人数: 1

C:\Users\zc\Desktop\Socket_Client.exe
——欢迎加入聊天!
有人吗? hello, everyone!
Client 1: 你也好, what's you name? (2023/10/21 18:08:46)
```

可见在服务端有离线的日志。

若服务端达到最大连接数, 会提示已达到最大连接:

```
——已达到最大连接数... ——Client 4 离线啦... (2023/10/21 18:11:15)
——在线人数: 4
——Client 1 离线啦... (2023/10/21 18:11:13)
——在线人数: 3
——Client 3 离线啦... (2023/10/21 18:11:14)
——在线人数: 2
——Client 2 离线啦... (2023/10/21 18:11:15)
——在线人数: 1
```

以上，程序完成了最初的方案实现。

四、总结

熟悉了 Socket 套接字网络编程。

熟悉了通信流程，消息交互的时序。

遇到的问题/思考：

对缓冲区消息的接收，以及相应的格式化，解决这个问题需要调用特定但不熟悉的函数，最后通过网络查阅解决。

本次程序体量较小，在现实中还需考虑大量客户端同时对服务端发起请求的问题。如负载均衡、水平扩展、缓存等等。

本次实验可以扩展的部分，对话界面、敏感词分析、客户端登录、私聊等等，属于优化但与本次实验目关联较小。