



南開大學  
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计算机网络实验报告

---

## 编程作业 1：聊天程序的设计与实现

---

聂志强 2012307

年级：2020 级

专业：信息安全

指导教师：徐敬东

2022 年 10 月 28 日

## 摘要

本次实验通过设计应用层聊天协议，利用流式 socket 调用传输层接口实现一个网络聊天程序。在双人通信中，通过建立两个线程进行通信；在多人聊天中，通过每个用户建立一个接收线程，服务器为每个用户建立一个线程实现通信。通过设定消息格式与解析算法实现定点或全体消息发送，创新的通过增加一个线程实现服务器端随时可以通过 exit 结束

**关键字：**多线程, Socket , 协议, Client, 分组转发

## 目录

一、 总体设计	1
(一) 实验要求	1
(二) 基本功能描述	1
(三) 实验环境	1
(四) 协议设计	2
1. 语法规则设计	2
2. 用户名域设计	2
3. 消息格式设计	2
4. 用户退出设计	2
二、 程序设计 (核心代码)	3
(一) 特色附加模块	3
1. 拆包	3
2. 用户名输入检验	4
(二) 多线程模块	4
1. 客户端接收线程	4
2. 服务器端处理连接线程	5
3. 服务器端消息接收与发送线程	6
(三) 常规架构模块	8
1. 时间标签	8
2. 服务器端用户信息保存	8
3. 客户端初始建立过程	9
4. 客户端发送消息	9
三、 运行效果	10
(一) 登录	10
(二) 群发	11
(三) 定点发送	12
(四) 退出	12
(五) 用户输入检验	13
(六) 回车检验	13

## 一、 总体设计

### (一) 实验要求

- 使用流式 Socket，设计一个两人聊天协议，要求聊天信息带有时间标签。请完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。
- 对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。
- 在 Windows 系统下，利用 C/C++ 对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。编写程序时，只能使用基本的 Socket 函数，不允许使用对 socket 封装后的类或架构。
- 对实现的程序进行测试。
- 正常的用户退出方式

### (二) 基本功能描述

- 支持双人聊天、多人群聊
- 支持群发或指定用户转发功能
- 用户登录验证功能
- 中英文聊天功能
- 用户和服务端合理性退出
- 宕机情况下服务器退出功能

### (三) 实验环境

实验平台：Visual Studio 2017

所需头文件：

头文件

```
1 #include <WinSock2.h>
2 #include <process.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <conio.h>
6 #include <iostream>
7 #include <time.h>
8 #include <string>
9 #pragma comment(lib, "Ws2_32.lib")
```

## (四) 协议设计

### 1. 语法规义设计

为了方便应用层与传输层之间的传输，我们设计消息的封装格式，即消息的语法。在该聊天室程序的两端 server 和 client 都遵循该聊天协议，符合协议的对等性。我们设计的完整消息长度为定长 128 bytes，其中第一个 byte 用于结束符，消息格式为 [SENDNAME : CONTENT]，即以冒号分隔接收者名称和消息具体内容。我们规定每个用户名称不能超过 9bytes，因此消息内容不能超过 118 bytes。

### 2. 用户名域设计

为支持不同的功能，用户名域有不同的格式：

- 定点发送：用户名域直接填写 SENDNAME
- 群发功能：用户名域可以填写 all 或者不填（不填默认群发）
- 用户名检查：由于 System（系统提示）以及 all（群发）已被我们占用，所以用户名称不能为 System 和 all；用户名还需满足我们所设计的大小

### 3. 消息格式设计

当用户需要指定向某个用户发送信息时，可以使用如下方法发送，服务器将识别第一个冒号来对信息进行切分。

[username] : [message]  
eg. Mary:hello!

当用户需要全局发送时，可以使用如下三种方法。

all:[message] eg. all:hello  
[message] eg. hello  
:[message] eg. :hello

当用户未输入任何内容仅输入回车发送时，不群发此消息但服务器日志会记录该行为。

### 4. 用户退出设计

客户端当检测到用户输入 exit 时，将退出当前线程并向服务器发送 exit，服务器端接收到客户端发送的 exit 请求，则不会转发 x 消息并直接关闭该线程

客户端

```
1 //如果用户输入exit准备退出
2 if (strcmp(bufferSend, "exit") == 0)
3 {
4     cout << "您已离开聊天室" << endl;
5     if (send(ClientSocket, bufferSend, sizeof(bufferSend), 0) ==
6         SOCKET_ERROR)
7         return -1;    //退出当前线程
8     break;
9 }
```

## 服务器端

```

1  if (strcmp(temp, "exit") == 0)    //
2  {
3      closesocket(inClient[flag].sClient); //关闭该套接字
4      CloseHandle(HandleRecv[flag]); //这里关闭了线程句柄
5      inClient[flag].sClient = 0; //把这个位置空出来，留给以后进入的线程使用
6      HandleRecv[flag] = NULL;
7      cout << "用户 [" << inClient[flag].userName << "]" << "离开聊天室 "
8      << endl;
9  }

```

## 二、 程序设计 (核心代码)

## (一) 特色附加模块

## 1. 拆包

服务器端接收到消息后，将消息按聊天协议格式进行解包，根据解析得到的消息结构做出相应的处理再进行转发。当识别到即以冒号分隔接收者名称和消息具体内容（如 [SENDNAME : CONTENT]），则对：左右两部分进行拆分对用户域进行解析后会判断执行定点发送或者群发功能，同时对内容进行解析，获得 real\_message 后进行封装，即以 [sendName]real\_message 的方式进行打包发送。

## 拆包

```

1  // 拆包：将消息按聊天协议的格式进行解包
2  memset(temp, 0, sizeof(temp));
3  if (recv(client, temp, sizeof(temp), 0) == SOCKET_ERROR)
4      continue;
5  string contents = temp;
6  sendname = contents.substr(0, contents.find(':'));
7  content = contents.substr(contents.find(':') + 1 == 0 ? contents.length() :
8      contents.find(':') + 1);
9  if (content.length() == 0)
10 {
11     strcpy(inClient[flag].identify, "all");
12     strcpy_s(temp, sendname.c_str());
13 }
14 else
15 {
16     strcpy(temp, content.c_str());
17     strcpy(inClient[flag].identify, sendname.c_str());
18 }
19 memcpy(inClient[flag].buffer, temp, sizeof(inClient[flag].buffer));

```

## 2. 用户名输入检验

由于涉及上述拆包以及避免“all”字符出现歧义，根据设计的协议中用户名规范，当输入用户名长度大于 9 或者为占用字符串“all”，系统提示重新输入直至输入合规再对服务器发送连接请求

### 用户名输入

```

1  bool st = true;
2  string name;
3  do{
4      if (st)
5          st = false;
6      else
7          cout << "您输入的用户名不合规，请重新输入：";
8      cin >> name;
9  } while (name.length() > 9 || name == "all");
10 strcpy_s(userName, name.c_str());

```

## (二) 多线程模块

### 1. 客户端接收线程

#### 用户退出设计

```

1  DWORD WINAPI handlerRequest(void* param)
2  {
3      char bufferRecv[128] = { 0 };
4      // 如果接收正确，则一直处于接收状态
5      while (true)
6      {
7          // 等待并接收消息
8          if (recv(*(SOCKET*)param, bufferRecv, sizeof(bufferRecv), 0)
9              == SOCKET_ERROR)
10             break;
11         if (strlen(bufferRecv) != 0)
12         {
13             // '\b' 光标迁移
14             // 坑！一定+2且带等号，有：和空格
15             for (int i = 0; i <= strlen(userName) + 40; i++)
16                 cout << "\b";
17             // 打印时间
18             time(&t);
19             strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
20             cout << '(' << str << ") ";
21             // 发送源的名字+数据内容
22             cout << bufferRecv << endl;
23             //// 因为这是在用户的send态时，把本来打印出来的
24             //      userName给退回去了，所以收到以后需要再把userName
25             //      打印出来

```

```

23         cout << '(' << str << ")  '[' << userName << "]" : ";
24     }
25 }
26 return 0;
27 }

```

## 2. 服务器端处理连接线程

### 用户退出设计

```

1  DWORD WINAPI Accept_thread(void* param)
2  {
3      int flag[10] = { 0 };
4      while (true)
5      {
6          if (inClient[i].flag != 0) //找到从前往后第一个没被连接的
            inClient
7          {
8              i++;
9              continue;
10         }
11         // accept()阻塞进程直到有客户端连接, 接受一个特定socket请求等待
            队列中的连接请求
12         if ((inClient[i].sClient = accept(ServerSocket, (SOCKADDR*)&
            ClientAddr, &ClientAddrLen)) == INVALID_SOCKET)
13         {
14             cout << "[System] Accept错误, 请通过 " <<
                WSAGetLastError() << "获取详情" << endl;
15             closesocket(ServerSocket);
16             WSACleanup();
17             return -1;
18         }
19         //接收用户名
20         recv(inClient[i].sClient, inClient[i].userName, sizeof(
            inClient[i].userName), 0);
21         cout << "[System] 客户端 [" << inClient[i].userName << "]" <<
            " 连接成功" << endl;
22         memcpy(inClient[i].IP, inet_ntoa(ClientAddr.sin_addr), sizeof(
            inClient[i].IP)); //记录客户端IP
23         inClient[i].flag = inClient[i].sClient; //不同的socket有不同
            UINT_PTR类型的数字来标识
24         i++;
25
26         //遍历其他客户端并创建进程
27         for (int j = 0; j < i; j++)
28         {
29             if (inClient[j].flag != flag[j])
30             {

```

```

31         if (HandleRecv[j])
32             CloseHandle(HandleRecv[j]);
33         //开启接收消息的线程
34         HandleRecv[j] = CreateThread(NULL, 0, (
            LPTHREAD_START_ROUTINE)Rec_Send_thread, &
            inClient[j].flag, 0, 0);
35     }
36 }
37 for (int j = 0; j < i; j++)
38     flag[j] = inClient[j].flag; //防止ThreadRecv线程多次开
        启
39     Sleep(3000);
40 }
41 return 0;
42 }

```

### 3. 服务器端消息接收与发送线程

通过调用 <time.h> 中的 time 函数获得发送消息时的时间并输出到命令行界面。

用户退出设计

```

1  DWORD WINAPI Rec_Send_thread(void* param)
2  {
3      bool Start = true;
4      SOCKET client = INVALID_SOCKET;
5      int flag = 0;
6      for (int j = 0; j < i; j++) {
7          if (*(int*)param == inClient[j].flag)
8              //判断是为哪个客户端开辟的接收数据线程
9              {
10                 client = inClient[j].sClient;
11                 flag = j;
12             }
13     }
14 }
15 char temp[128] = { 0 }; //临时数据缓冲区
16 string sendname, content;
17 while (true)
18 {
19     //拆包, 解析发送消息范围
20     memset(temp, 0, sizeof(temp));
21     if (recv(client, temp, sizeof(temp), 0) == SOCKET_ERROR)
22         continue;
23     string contents = temp;
24     sendname = contents.substr(0, contents.find(':'));
25     content = contents.substr(contents.find(':') + 1 == 0 ? contents.
        length() : contents.find(':') + 1);
26     if (content.length() == 0)

```



```

27     {
28         strcpy(inClient[flag].identify, "all");
29         strcpy_s(temp, sendname.c_str());
30     }
31     else
32     {
33         strcpy(temp, content.c_str());
34         strcpy(inClient[flag].identify, sendname.c_str());
35     }
36     memcpy(inClient[flag].buffer, temp, sizeof(inClient[flag].buffer));
37
38     //判断如果客户发送exit请求, 那么直接关闭线程, 不打开转发线程
39     if (strcmp(temp, "exit") == 0)
40     {
41         closesocket(inClient[flag].sClient); //关闭该套接字
42         CloseHandle(HandleRecv[flag]); //这里关闭了线程句柄
43         inClient[flag].sClient = 0; //把这个位置空出来, 留给以后进入
           的线程使用
44         HandleRecv[flag] = NULL;
45         cout << "[System] 用户 [" << inClient[flag].userName << "] "
           << "离开聊天室 " << endl;
46     }
47     else if (Start == true)
48     {
49         Start = false;
50         continue;
51     }
52     else
53     {
54         time(&t);
55         strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
56         //cout << str << endl;
57         cout << '(' << str << ") [" << inClient[flag].userName << "]
           : " << temp << endl;
58         char temp[128] = { 0 }; //创建一个临时的数据缓冲
           区, 用来存放接收到的数据
59         memcpy(temp, inClient[flag].buffer, sizeof(temp));
60         sprintf(inClient[flag].buffer, "%s: %s", inClient[flag].
           userName, temp); //把发送源的名字添进转发的信息里
61         if (strlen(temp) != 0) //如果数据不为空且还没转发则转发
62         {
63             // 向所有用户发送
64             if (strcmp(inClient[flag].identify, "all") == 0)
65             {
66                 for (int j = 0; j < i; j++)
67                     if (j != flag)
68                         //向除自己之外的所有客户端发
                           送信息

```

```

69         if (send(inClient[j].sClient,
70                 inClient[flag].buffer,
71                 sizeof(inClient[j].buffer), 0) == SOCKET_ERROR)
72             return -1;
73     }
74     else
75         // 向指定用户发送
76         for (int j = 0; j < i; j++)
77             if (strcmp(inClient[j].userName,
78                     inClient[flag].identify) == 0)
79                 if (send(inClient[j].sClient,
80                         inClient[flag].buffer,
81                         sizeof(inClient[j].buffer), 0) == SOCKET_ERROR)
82                     return 1;
83     }
84 }
85 return 0;
86 }

```

### (三) 常规架构模块

#### 1. 时间标签

通过调用 <time.h> 中的 time 函数获得发送消息时的时间并输出到命令行界面。

用户退出设计

```

1 // time函数获取从1970年1月1日0时0分0秒到此时的秒数
2 time(&t);
3 //将时间格式化
4 strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
5 cout << str << endl;

```

#### 2. 服务器端用户信息保存

通过结构体数组保存客户端信息：套接字、数据、用户名、转发范围、IP 地址、标记

客户端信息保存

```

1 struct Client
2 {
3     SOCKET sClient;    //客户端套接字
4     char buffer[128];  //数据缓冲区
5     char userName[10]; //客户端用户名
6     char identify[16]; //用于标识转发的范围
7     char IP[20];       //客户端IP
8     UINT_PTR flag;     //标记客户端，用来区分不同的客户端

```

```
9 } inClient[10]; //创建一个客户端结构体,最多同时10人在线
```

### 3. 客户端初始建立过程

步骤: 初始化套接字 -> 创建套接字 -> 请求连接服务器

在设定所连接服务器端口号时需要参照服务器设置的端口号一致, IP 地址用本机 IP 地址 127.0.0.1 即可。

#### 服务器 IP 以及端口号

```
1 //定义获得可用socket的详细信息的变量,存放被WSAStartup函数调用后返回的Windows
  Sockets数据的数据结构
2 if (WSAStartup(MAKEWORD(2, 2), &wsaData))
3 {
4     cout << "[System] WSAStartup创建失败, 请通过" << WSAGetLastError() <<
      "获取详情" << endl;
5     return -1;
6 }
7
8 //创建套接字
9 ClientSocket = socket(AF_INET, SOCK_STREAM, 0);
10 if (ClientSocket == INVALID_SOCKET)
11 {
12     cout << "[System] Socket创建错误, 请通过" << WSAGetLastError() << "获
      取详情" << endl;
13     return -1;
14 }
15
16 //设置所连接服务器地址
17 ServerAddr.sin_family = AF_INET;
18 ServerAddr.sin_port = htons(uPort); //客户端端口号
19 ServerAddr.sin_addr.S_un.S_addr = inet_addr(IP); //客户端IP地址
20
21 cout << "[System] 正在连接..." << endl;
22
23 //连接服务器
24 if (SOCKET_ERROR == connect(ClientSocket, (SOCKADDR*)&ServerAddr, sizeof(
      ServerAddr)))
25 {
26     cout << "[System] Connect连接错误, 请通过" << WSAGetLastError() << "
      获取详情" << endl;
27     closesocket(ClientSocket);
28     WSACleanup();
29     return -1;
30 }
```

### 4. 客户端发送消息

## 用户退出设计

```
1 while (true)
2 {
3     if (start)
4     {
5         // time函数获取从1970年1月1日0时0分0秒到此时的秒数
6         time(&t);
7         //将时间格式化
8         strftime(str, 20, "%Y-%m-%d %X", localtime(&t));
9         //cout << str << endl;
10        cout << '(' << str << ") ";
11        cout << '[' << userName << "]" : ";
12    }
13    start = true;
14    // 用户输入发送消息
15    cin.getline(bufferSend, 128);
16    //如果用户输入exit准备退出
17    if (strcmp(bufferSend, "exit") == 0)
18    {
19        cout << "您已离开聊天室" << endl;
20        CloseHandle(handlerRequest);
21        if (send(ClientSocket, bufferSend, sizeof(bufferSend), 0) ==
22            SOCKET_ERROR)
23            return -1; //退出当前线程
24        break;
25    }
26    send(ClientSocket, bufferSend, sizeof(bufferSend), 0);
27 }
```

## 三、 运行效果

## (一) 登录

先开启服务器端等待用户连接，Bib、Alice、Tim 输入用户名并检查合规后依次进入聊天室，服务器端记录日志（如红框所示）

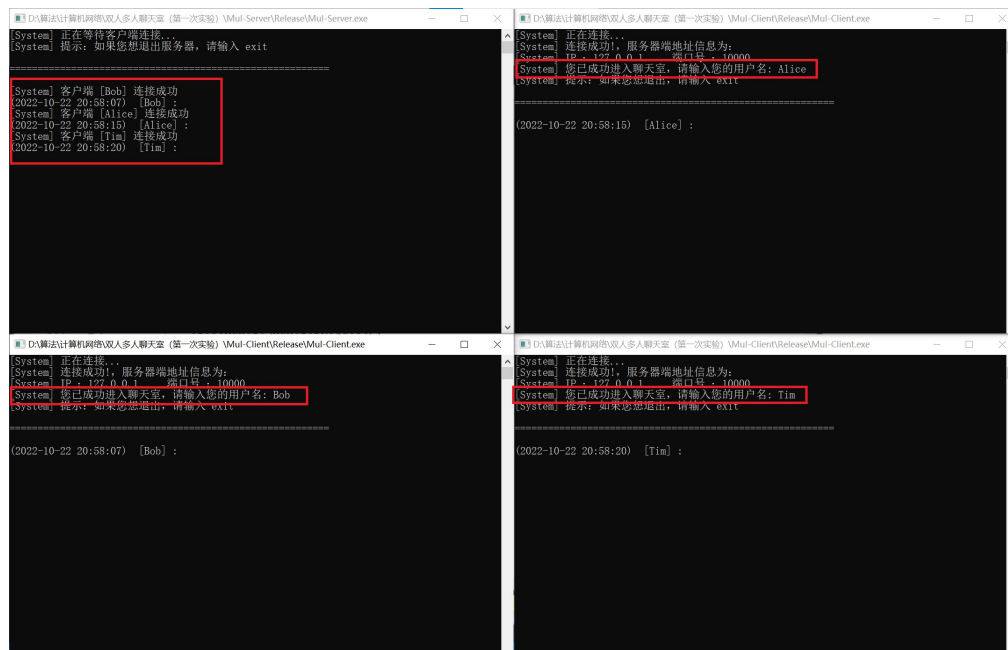


图 1: 多人聊天-用户登录

## (二) 群发

根据协议，当用户输入的消息满足三种格式 (all:[message] 或 [message] 或:[message]) 之一时，即为向其他所有客户端转发该消息内容字段，如下图所示，当用户 Bob 和 Alice 直接输入“大家好”和“Nice to meet you”时，系统识别为群发消息，因此将这两条消息转发给所有客户端并在服务器端保存该日志。

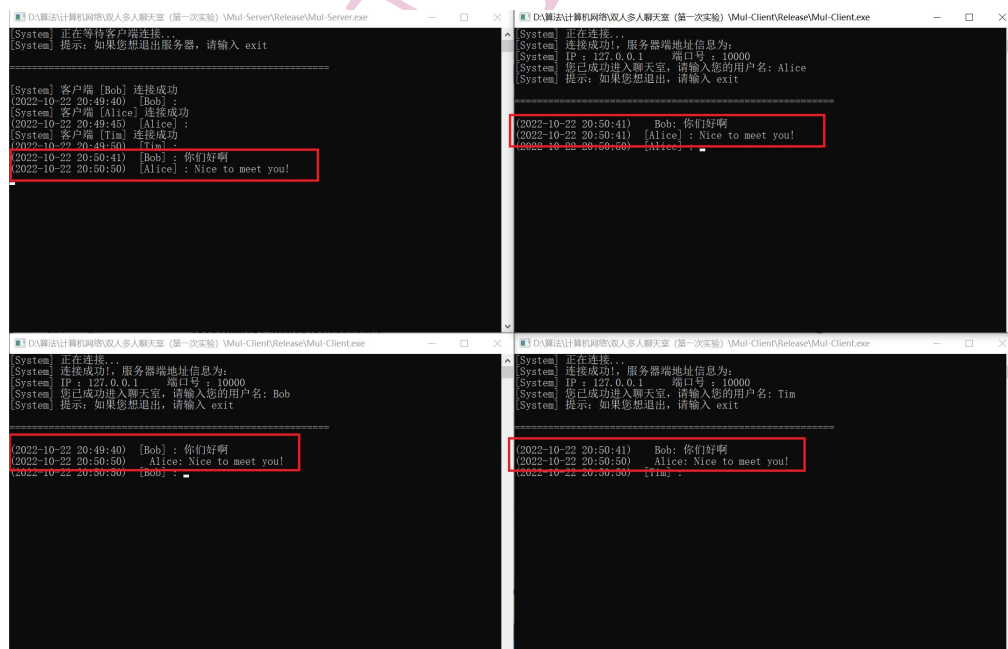
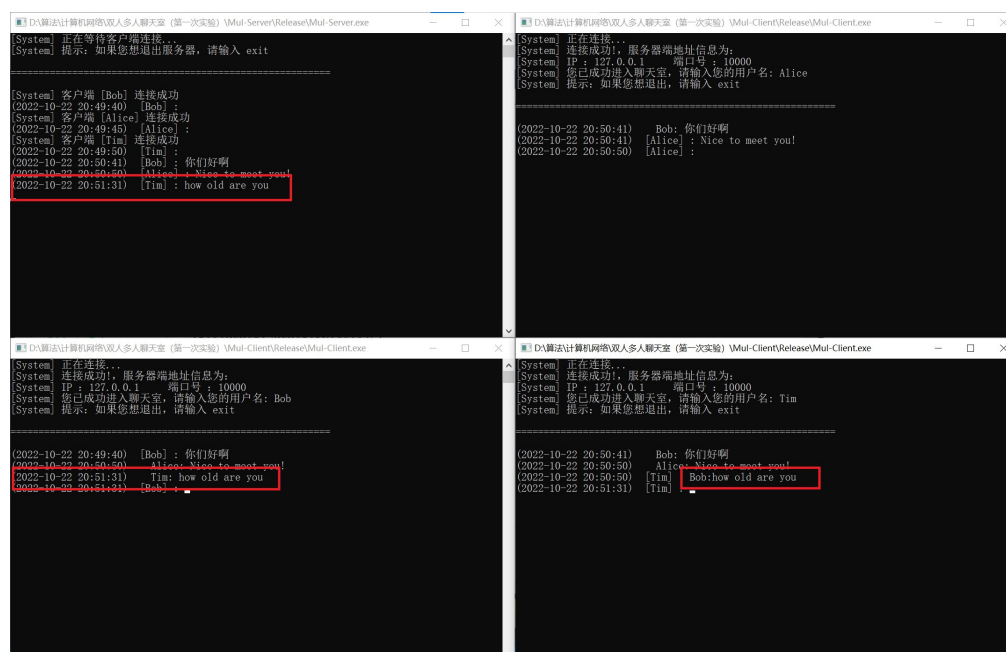


图 2: 多人聊天-群发

### (三) 定点发送

根据协议，用户输入的消息以‘:’分隔时，‘:’前的字符串为该消息指定转发的对象，并记录至服务器日志，如下图所示，当 Tim 输入”Bob:how old are you” 时，表明该条消息由 Tim 单独指定转发给 Bob，此时可以观察到 Bob 客户端出现该消息并在服务器日志记录，Alice 客户端并未出现该消息。



```
System] 正在等待客户端连接...
[System] 提示: 如果您想退出服务器, 请输入 exit

[System] 客户端 [Bob] 连接成功
(2022-10-22 20:49:40) [Bob]:
[System] 客户端 [Alice] 连接成功
(2022-10-22 20:49:45) [Alice]:
[System] 客户端 [Tim] 连接成功
(2022-10-22 20:49:50) [Tim]:
(2022-10-22 20:50:41) [Bob]: 你们好啊
(2022-10-22 20:50:41) [Alice]: Nice to meet you!
(2022-10-22 20:50:50) [Tim]: Bob:how old are you
(2022-10-22 20:51:31) [Tim]:

System] 正在连接...
[System] 连接成功, 服务器端地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Alice
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:50:41) [Bob]: 你们好啊
(2022-10-22 20:50:41) [Alice]: Nice to meet you!
(2022-10-22 20:50:50) [Alice]:
(2022-10-22 20:50:50) [Tim]: Bob:how old are you
(2022-10-22 20:51:31) [Tim]:

System] 正在连接...
[System] 连接成功, 服务器端地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Bob
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:49:40) [Bob]: 你们好啊
(2022-10-22 20:50:50) [Alice]: Nice to meet you!
(2022-10-22 20:51:31) [Tim]: how old are you
(2022-10-22 20:51:31) [Bob]:

System] 正在连接...
[System] 连接成功, 服务器端地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Tim
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:50:41) [Bob]: 你们好啊
(2022-10-22 20:50:50) [Alice]: Nice to meet you!
(2022-10-22 20:50:50) [Tim]: Bob:how old are you
(2022-10-22 20:51:31) [Tim]:
```

图 3: 多人聊天-定点发送

### (四) 退出

当客户端输入 exit 时, 该用户退出并在服务器端记录日志, 考虑实际情况, 当服务器宕机等情况出现时, 我们通过在服务器端输入 exit 也可以使服务器直接退出, 如下图所示, Alice、Tim、Bob 依次离开聊天室, 服务器记录三条日志, 最后在服务器端输入 exit, 服务器正常退出

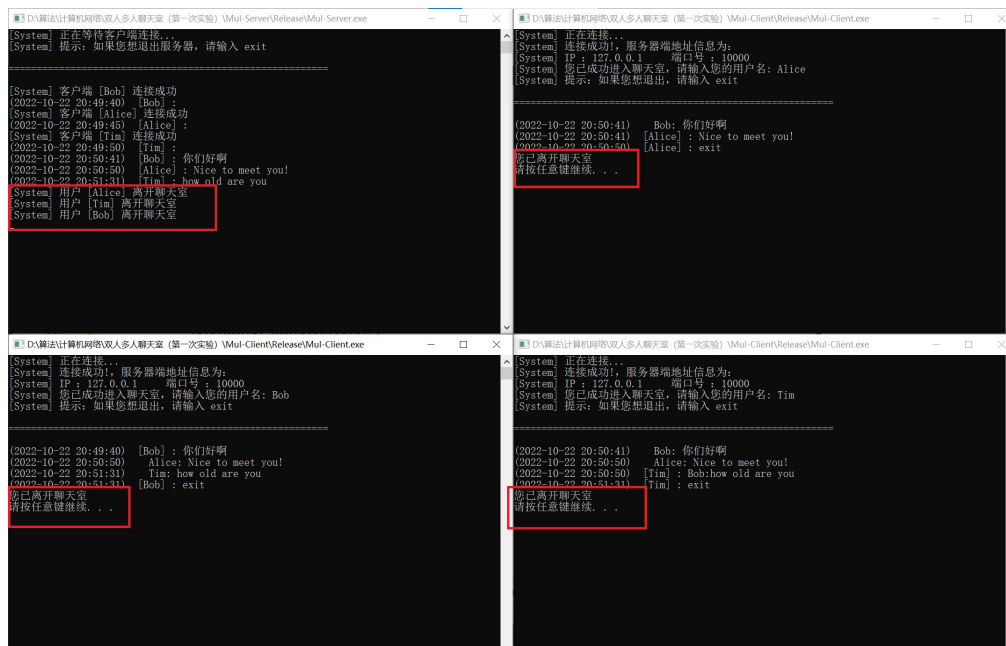


图 4: 客户端与服务端退出

## (五) 用户输入检验

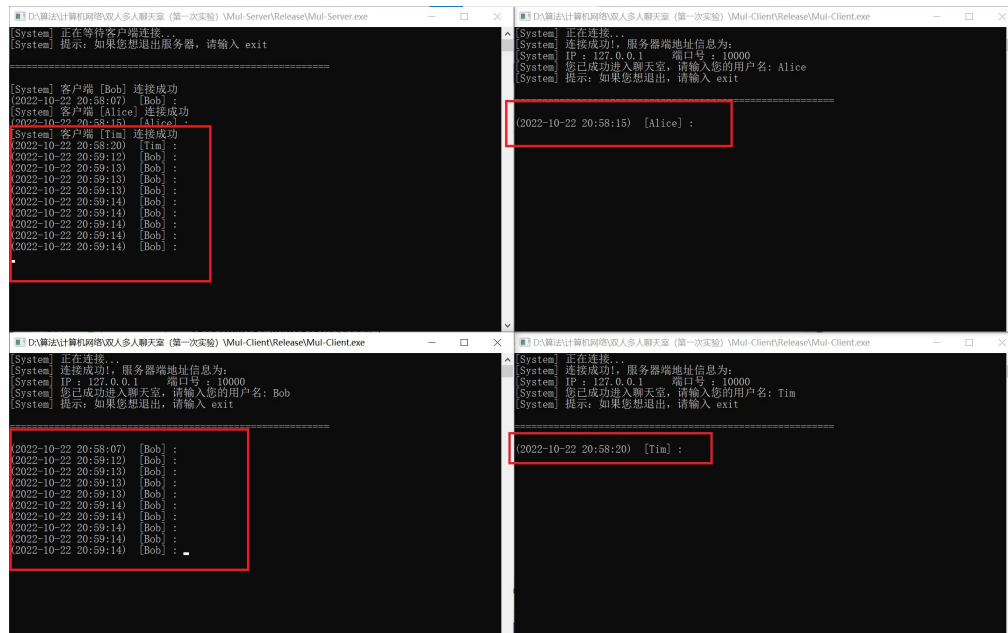
根据协议内容，输入用户名不能超过 9 位，如下图所示，当用户名输入不合法时系统提示重新输入直至合规并将用户名发送至服务器端。



图 5: 用户名输入检验

## (六) 回車檢驗

根据协议规则，当用户仅输入回车发送消息时，该消息不发送给其他客户端，但会在服务器日记记录该行为。如下图所示，Bob 输入多个回车，此时 Alice 和 Tim 客户端并没有该响应，服务器日志记录 Bob 该行为：



```
[System] 正在等待客户端连接...
[System] 提示: 如果您想退出服务器, 请输入 exit

[System] 客户端 [Bob] 连接成功
(2022-10-22 20:58:07) [Bob] :
[System] 客户端 [Alice] 连接成功
(2022-10-22 20:58:15) [Alice] :
[System] 客户端 [Tim] 连接成功
(2022-10-22 20:58:20) [Tim] :
(2022-10-22 20:59:12) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :

[System] 正在连接...
[System] 连接成功!, 服务器地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Alice
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:58:15) [Alice] :

[System] 正在连接...
[System] 连接成功!, 服务器地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Bob
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:58:07) [Bob] :
(2022-10-22 20:59:12) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:13) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :
(2022-10-22 20:59:14) [Bob] :

[System] 正在连接...
[System] 连接成功!, 服务器地址信息为:
[System] IP: 127.0.0.1 端口号: 10000
[System] 您已成功进入聊天室, 请输入您的用户名: Tim
[System] 提示: 如果您想退出, 请输入 exit

(2022-10-22 20:58:20) [Tim] :
```

图 6: 回车检验