



南開大學

Nankai University

计算机学院
计算机网络实验报告

Socket 编程

姓名：杨冰雪

学号：2110508

专业：计算机科学与技术

2023 年 10 月 20 日

目录

1 实验要求	2
2 实验模块	2
2.1 服务端	2
2.1.1 主线程	2
2.1.2 子线程	2
2.2 客户端	3
2.2.1 主线程	3
2.2.2 子线程	3
3 协议设计	3
3.1 语法	4
3.2 语义	4
3.3 时序	4
4 核心代码讲解	4
4.1 服务端广播函数	4
4.2 服务端接收转发函数	5
4.3 客户端发送消息函数	6
4.4 客户端接收消息函数	6
5 运行结果	7
6 实验总结	8

1 实验要求

- 利用 C 或 C++ 语言，使用基本的 Socket 函数、流式套接字、采用多线程（或多进程）方式完成一个支持多人聊天的程序。
- 要求程序能够支持英文和中文聊天，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。
- 编写的程序应该结构清晰，具有较好的可读性。

2 实验模块

本实验的应用场景主要为多人聊天，所以主要角色可以分为服务端和客户端。运行时应该只有一个服务端，存在多个客户端连接服务端，通过服务端的转发实现多人聊天的功能。

2.1 服务端

为了能够让服务端能够连接多个客户端，所以我们采用了多线程，其主线程主要用于当有客户端请求连接时接受连接，同时为了使服务端能够边发送边接收消息，我们把发送消息和接收消息的功能分离开，用不同的子线程去运行。

2.1.1 主线程

- 打开网络库——`open_socket(WSAStartup)`
- 创建并初始化服务端 Socket，监听客户端连接——`create_server_socket(socket,bind,listen)`
- 创建广播消息线程向所有用户广播消息——`serv_send`。
- 进入 `while(true)` 循环，阻塞于 `accept` 函数
- 当有客户端请求连接时接受连接，记录客户端的 IP 地址和端口号和名字，单独为该用户创建接收转发线程——`serv_recv`。
- 主线程回到 `accept` 函数处继续等待用户端连接。

2.1.2 子线程

子线程中只需要一个广播消息线程就可以把服务器的消息广播到各个子线程，但接受转发线程则对于每个用户都要创建个子线程，这样才能接收每个用户的消息，然后把消息转发给其他用户。

接收转发线程

- 进入 `while(true)` 循环，开始接收客户端消息并阻塞——`recv`。
- 当接收到客户端消息后，把接收的消息与结束消息比较。
 - 如果相同则关闭客户端，结束该进程，代表用户退出聊天——`closesocket`。
 - 如果不相同，则接收用户发出的消息，并向所有用户广播该消息——`send`。

广播消息线程

- 进入 `while(true)` 循环，开始接收服务端输入消息并阻塞——`cin.getline`。
- 当服务端输入了消息后，判断连接服务端的用户数量。
 - 当用户数量为 0，表示没有用户连接，则不进行广播。
 - 当用户数量不为 0，表示存在用户连接，向所有用户发送消息，对于没有按照规定退出的用户，向服务端发出消息提示，并关闭该客户端——`send`、`closesocket`。

2.2 客户端

为了能让客户端边接收信息边发送信息，所以在客户端中创建了两个子线程，一个用于接收消息，一个用于发送消息。

2.2.1 主线程

- 打开网络库——`open_Socket(WSAStartup)`
- 创建客户端 Socket 并向服务器发起连接——`create_ClientSocket(socket,connect)`。
- 向服务端发送名字——`send`。
- 创建子线程用于发送和接收消息并阻塞于线程回收函数，等待回收发送消息线程。
- 当发送消息线程被回收后，关闭客户端和网络库——`closesocket`、`close_Socket(WSACleanup)`。

2.2.2 子线程

发送信息线程

- 进入 `while(true)` 循环，等待用户输入信息而阻塞——`cin.getline`。
- 将用户输入的消息发送给服务端——`send`。
- 判断用户发送消息与聊天结束消息是否相同——`strcmp`。
 - 当相同时，线程退出执行函数
 - 当不相同，线程继续执行循环

接收信息线程

- 进入 `while(true)` 循环，接收服务端的消息而阻塞——`recv`。
- 当接收到服务端的消息时，打印消息到屏幕上。
- 继续执行循环。

3 协议设计

本次实验，使用 Client 和 Server 端进行通信。在该聊天程序的两端 server 和 client 遵循如下聊天协议。

3.1 语法

报文包括三种类型的数据，分别是发送消息类型、接收消息类型和发送内容类型。发送消息和接收消息的报文都设了大小限制，不得超过 500 字节；而发送内容类型是服务器接收到客户端消息后向其他用户发送的报文，它包含了用户的名字和用户发送的消息内容，因为设置的用户名字大小不得超过 20 字节，用户发送的消息内容大小不得超过 500 字节，再加上字符:，所以报文长度限制为 521 字节。通过向用户发送内容中包含用户姓名，而不是让用户直接可以获取用户姓名，这样能保证用户无法获取用户姓名，从而保障了用户信息的安全。

3.2 语义

由于为了支持中文消息的发送，我们本次所传输的报文都统一采用 GBK 编码方式，此外消息都是通过换行符进行消息的截断的。

3.3 时序

首先运行 service.cpp，建立 socket 并进行初始化，执行 bind 函数绑定 IP 地址和端口号，让其进入监听状态。在等待客户端的连接请求时，一旦收到客户端的连接请求时，就会执行 accept 函数接收连接。然后继续监听客户端的连接，从而能够连接多个客户端。并且采取多线程的方式，可以使每一个客户端在任意时刻发送任意数量的消息，同时可以顺利接收其他人的消息，从而实现较为合理的聊天功能。

4 核心代码讲解

4.1 服务端广播函数

serv_send

```
1 void* serv_send(void* arg){
2     while (1){
3         //向所有客户端发送消息
4         char content[MAX_CONTENT_LEN] = { 0 };
5         strcpy(content, "server:");
6         char sendbuf[MAX_SEND_LEN];
7         cin.getline(sendbuf, MAX_SEND_LEN);
8         strcat(content, sendbuf);
9         if (All_Clientfd.size() == 0){
10             cout << "[提示]: 没有连接的客户端" << endl;
11         }
12         else {
13             for (int i = 0; i < All_Clientfd.size(); i++){
14                 if (send(All_Clientfd[i], content, MAX_CONTENT_LEN, 0) == SOCKET_ERROR){
15                     cout << "service send failed:" << WSAGetLastError << endl;
16                     cout << "[信息]: 可能存在客户已退出聊天室!" << endl;
17                     closesocket(All_Clientfd[i]);
18                     All_Clientfd.erase(All_Clientfd.begin()+i);
19                 }
20             }
21             return NULL;
22         }
23     }
24 }
```

```

20     }
21 }
22 }
23 return NULL;
24 }

```

- 用 `sendbuf` 字符型数组来储存服务端广播消息的内容，数组大小为 500 字节，表示其广播内容大小不能超过 500 字节。
- 服务端使用 `cin.getline` 函数来输入消息，使得消息内容中能包括空格等符号，通过换行符截取消息内容。
- `All_Clientfd` 是一个 `SOCKET` 类型的 `vector`，储存了连接服务端的所有客户端的 `socket`。
- 如果 `All_Clientfd` 大小为 0，则表示没有客户端连接，否则向所有客户端发送消息。
- 对于发送消息失败的客户端，自动断开连接，关闭客户端。

4.2 服务端接收转发函数

serv_recv

```

1 void* serv_recv(void* arg){
2     serv_thread* server = (serv_thread*)arg;
3     const SOCKET clientfd = server->clientfd;
4     char name[MAX_NAME_LEN] = {0};
5     strcpy(name, server->name);
6     char endbuf[MAX_REC_LEN] = { 0 };
7     strcpy(endbuf,ENDSIGNAL);
8     while (1){
9         char recbuf[MAX_REC_LEN] = { 0 };
10        //接收客户端消息
11        if (recv(clientfd, recbuf, MAX_REC_LEN, NULL) > 0){
12            if (strcmp(recbuf, endbuf) == 0){
13                cout << "[提示]: "<<name<<" 已经退出聊天!" << endl;
14                closesocket(clientfd);
15                All_Clientfd.erase(remove(All_Clientfd.begin(), All_Clientfd.end(),
16                    clientfd), All_Clientfd.end());
17                return NULL;
18            }
19            cout << name << ": " << recbuf << endl;
20            //向所有客户端广播消息
21            char content[MAX_CONTENT_LEN] = { 0 };
22            strcpy(content, name);
23            strcat(content, ":");
24            strcat(content, recbuf);
25            for (int i = 0;i < All_Clientfd.size();i++){
26                if (send(All_Clientfd[i], content, MAX_CONTENT_LEN, NULL) == SOCKET_ERROR){
27                    cout << "broadcast failed:" << WSAGetLastError << endl;

```

```

27     }
28     }
29     }
30 }
31 return NULL;
32 }

```

- 定义一个字符型数组 endbuf，用来储存结束退出符号。
- 接收用户端消息，使用 strcmp 函数把接收消息和结束符号进行比较，当相同时，表示用户退出聊天，关闭该用户端。
- 如果不相同，则向所有用户转发该消息，从而实现多人聊天。

4.3 客户端发送消息函数

client_send

```

1 void* client_send(void* arg){
2     client_thread* client = (client_thread*)arg;
3     char endbuf[MAX_NAME_LEN] = { 0 };
4     strcpy(endbuf, ENDSIGNAL);
5     while (1){
6         // 发送消息给服务端
7         char sendbuf[MAX_SEND_LEN] = { 0 };
8         cin.getline(sendbuf, MAX_SEND_LEN);
9         if (send(client->clientfd, sendbuf, MAX_SEND_LEN, NULL) == SOCKET_ERROR){
10             cout << "client send failed:" << WSAGetLastError << endl;
11         }
12         if (strcmp(sendbuf, endbuf) == 0){
13             return NULL;
14         }
15     }
16     return NULL;
17 }

```

- 定义一个字符型数组 endbuf，用来存储结束退出符号。
- 使用 cin.getline 函数输入用户想要发送的消息，以换行符截取消息。
- 向服务端发送客户端输入的消息。
- 调用 strcmp 函数把客户端输入的消息与结束符号进行比较，如果相同则进行函数返回。

4.4 客户端接收消息函数

client_recv

```

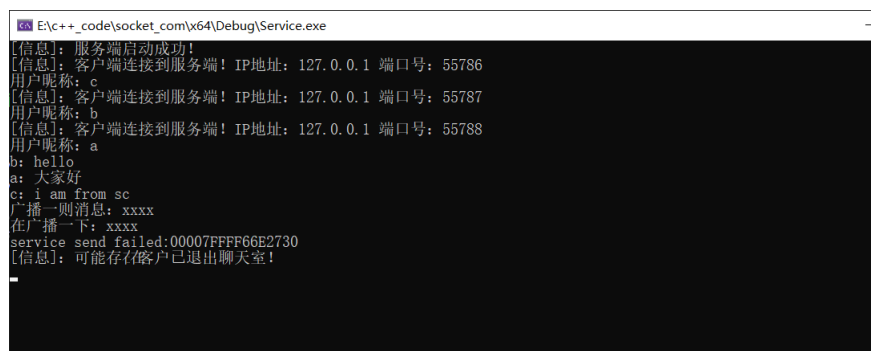
1 void* client_recv(void* arg){

```

```
2 client_thread* client = (client_thread*)arg;
3 while (1){
4     //接收服务端的消息
5     char recvbuf[MAX_REC_LEN] = { 0 };
6     if (recv(client->clientfd, recvbuf, MAX_REC_LEN, NULL) > 0){
7         cout << recvbuf << endl;
8     }
9 }
10 return NULL;
11 }
```

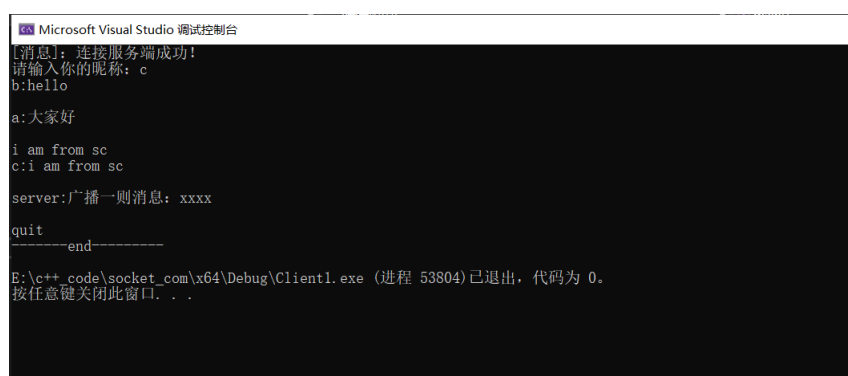
- 定义一个字符型数组 recvbuf 用来储存用户端发送的消息。
- 接收用户端发送的消息，调用 cout 在屏幕上输出打印出来。

5 运行结果



```
E:\c++_code\socket_com\x64\Debug\Service.exe
[信息]: 服务端启动成功!
[信息]: 客户端连接到服务端! IP地址: 127.0.0.1 端口号: 55786
用户昵称: c
[信息]: 客户端连接到服务端! IP地址: 127.0.0.1 端口号: 55787
用户昵称: b
[信息]: 客户端连接到服务端! IP地址: 127.0.0.1 端口号: 55788
用户昵称: a
b: hello
a: 大家好
c: i am from sc
广播一则消息: xxxx
在广播一下: xxxx
service send failed:00007FFFF66E2730
[信息]: 可能存在客户已退出聊天室!
```

图 5.1: 服务端



```
Microsoft Visual Studio 调试控制台
[消息]: 连接服务端成功!
请输入你的昵称: c
b:hello

a:大家好

i am from sc
c:i am from sc

server:广播一则消息: xxxx

quit
-----end-----

E:\c++_code\socket_com\x64\Debug\Client1.exe (进程 53804) 已退出, 代码为 0。
按任意键关闭此窗口。 . .
```

图 5.2: 客户端 1



```
Microsoft Visual Studio 调试控制台
[消息]: 连接服务端成功!
请输入你的昵称: b
hello
b:hello

a:大家好

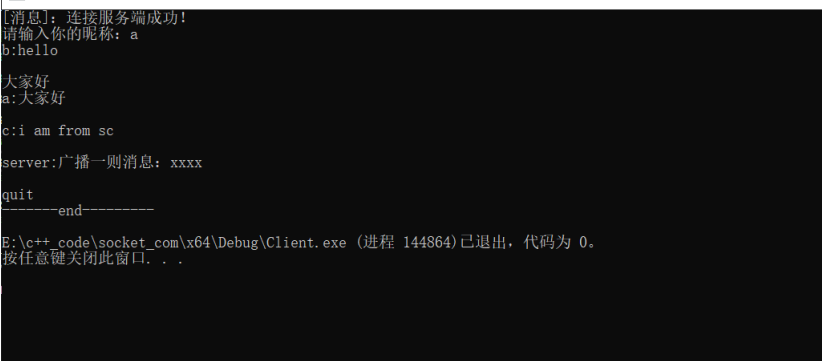
c:i am from sc

server:广播一则消息: xxxx

quit
-----end-----

E:\c++_code\socket_com\x64\Debug\Client2.exe (进程 145516) 已退出, 代码为 0。
按任意键关闭此窗口。 . .
```

图 5.3: 客户端 2



```
Microsoft Visual Studio 调试控制台
[消息]: 连接服务端成功!
请输入你的昵称: a
b:hello

大家好
a:大家好

c:i am from sc

server:广播一则消息: xxxx

quit
-----end-----

E:\c++_code\socket_com\x64\Debug\Client.exe (进程 144864) 已退出, 代码为 0。
按任意键关闭此窗口。 . .
```

图 5.4: 客户端 3

6 实验总结

通过本次实验,对 tcp 和 ip 协议有了更深刻的认识和理解,学会了使用 socket 进行编程来建立服务端和客户端的连接,也学会了调用多线程来实现多人聊天和边发消息边接收消息,但实验编程过程中仍然存在着一些问题,以下我将简要概况并说明我解决的方法。

1. 线程参数问题描述

因为在服务端中,我为每个用户都创建了一个接收转发线程,把需要传入的参数——用户端的 socket、用户姓名、用户 IP 地址、用户端口号等打包在一个结构体中,变成 void* 类型作为传入参数,但因为传入的是指针,所以每次经过一次循环后,其指针指向的值都为自动变成最后一个连接的用户的消息接收转发。

2. 解决方案

通过在线程里面创建常量记录下用户的信息,再作为参数进行使用,而不是直接使用函数传入的参数,这样参数的变化对函数的运行也不会有任何影响,从而能够实现向所有用户进行消息接收转发。