



南開大學
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

利用 Socket，编写聊天程序

姓名：唐明昊

学号：2113927

年级：2021 级

专业：计算机科学与技术

指导教师：吴英

2023 年 10 月 20 日

目 录

1 协议设计	2
2 模块功能	3
2.1 Server	3
2.1.1 Accept 模块	3
2.1.2 Receive 模块	4
2.1.3 Send 模块	5
2.2 Client	6
2.2.1 Send 模块	6
2.2.2 Receive 模块	7
3 运行展示	8
3.1 程序启动	8
3.2 消息发送	8
3.3 程序退出	9
3.4 再次上线	9
3.5 日志记录	9
4 实验总结	10
4.1 实验遇到的问题	10
4.2 实验思考	10

1 协议设计

实验采用的传输层协议为 TCP 协议，使用流式套接字传输数据。

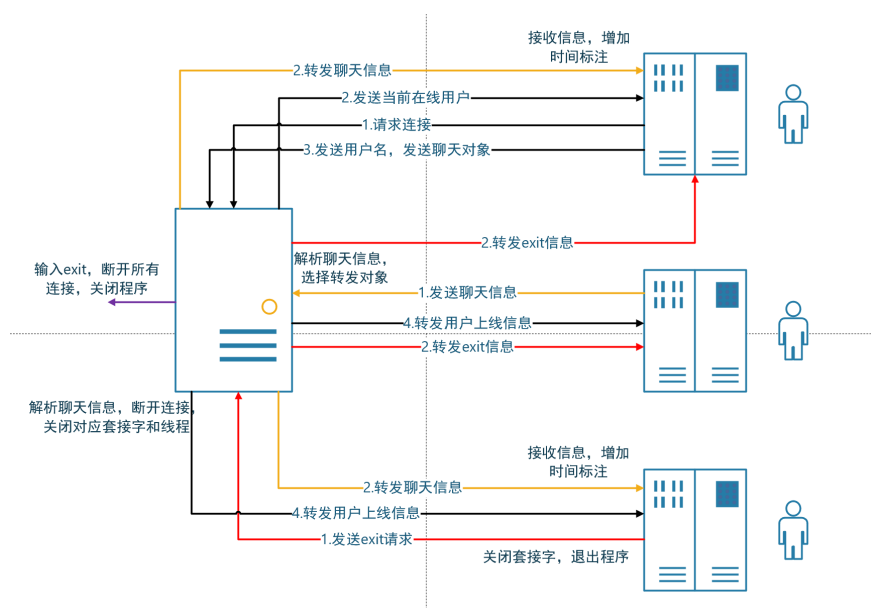


图 1.1: 整体示意图

应用层设计为一个核心 Server 端负责接收转发 Client 端的信息：多个 Client 端连接到 Server 端，Client 端选择发送对象，Server 端进行转发。

具体来说，聊天室程序支持多人（多个 Client）同时在线，但需要都连接到 Server 端进行管理。

- Client 端与 Server 端连接时，Server 端向 Client 发送当前聊天室中的在线用户，以便后续让 Client 选择聊天。
- Client 发送用户名给 Server 端进行记录以便后续管理转发消息等。
- Client 可以选择特定用户（向 Server 发送特定用户名）；也可以选择群发自己的信息（向 Server 发送“all”）。
- Client 端上线完毕，Server 端将告诉其他线程该用户上线。

完成上述流程后，Server 会为单个 Client 开辟一个数据接收线程，负责接收 Client 端的信息并进行转发。

- 如果客户端发送的是退出请求“exit”，Server 端会关闭对应线程和套接字，输出系统提示信息，并将该用户下线的提示信息转发给其他用户。客户端在发送该条请求后也会退出程序。
- 否则，Server 会再开辟一个线程用来转发消息，并打印该条消息记录（时间、用户名、内容与接收对象）。

消息转发线程会在消息前添加发送人的用户名，并根据连接时用户指定的对象转发消息。

Client 也采用的多线程模式，让发送与接收分开执行。当 Server 向某个 Client 转发消息时，Client 的接收消息线程截获消息，打印时间和从 Server 端得到的消息（用户名及内容）。

2 模块功能

2.1 Server

2.1.1 Accept 模块

Accept 线程负责接收 client 端的连接请求，接收到请求后，将 client 端的 socket 和用户名记录到结构体中，然后开辟一个线程负责接收转发 client 端发送的消息。

1. 按照协议，server 端首先接收刚连接上的 client 的用户名
2. 接着遍历 Client 结构体中找到所有在线的 client，将这些 client 的用户名发送给刚连接上的 client
3. 接收用户希望聊天的对象，并设置对应的结构体字段
4. 记录客户端 IP 等信息，为该 client 开辟消息转发线程，持续监听从客户端发来的消息并转发到对应的 client 端

值得一提的是，Accept 线程是在 THREAD_NUM 个线程中循环寻址未连接的线程，只有当前线程未连接或已断开连接才会进行后续 accept 等待连接。

```
// 接受请求
unsigned __stdcall ThreadAccept(void* param)
{
    int i = 0;
    while (true)
    {
        if (Clients[i].flag != 0) // 寻找未连接的线程
        {
            i++;
            i %= THREAD_NUM;
            continue;
        }

        // 等待连接，如果有客户端申请连接就接受连接
        if ((Clients[i].sClient = accept(ServerSocket, (SOCKADDR*)&ClientAddr, &ClientAddrLen)) ==
            INVALID_SOCKET)
            ...

        // 告诉用户目前有哪些人
        char tempNameBuffer[300] = "At present these people are in the chat room: ";
        for (int j = 0; j < NowClient; j++) {
            if(Clients[j].flag != 0)
                sprintf_s(tempNameBuffer, "%s \n [%s]", tempNameBuffer, Clients[j].userName); //
                格式化：把发送源的名字添加进信息里
        }
        send(Clients[i].sClient, tempNameBuffer, sizeof(tempNameBuffer), 0);

        recv(Clients[i].sClient, Clients[i].userName, sizeof(Clients[i].userName), 0); // 接收用户名
        recv(Clients[i].sClient, Clients[i].transID, sizeof(Clients[i].transID), 0); //
        接收转发的范围
        // 打印系统提示
```

```

...

Clients[i].flag = Clients[i].sClient; // 不同的socket由不同UINT_PTR类型的数字来标识
HandleRecv[i] = (HANDLE)_beginthreadex(NULL, 0, ThreadRecv, &Clients[i].flag, 0, NULL); //
    开启接收消息的线程

// 告诉其他用户有人上线
...
for (int j = 0; j < THREAD_NUM; j++) {
    if(Clients[j].flag != 0 && j!=i)
        send(Clients[j].sClient, tempMessage, sizeof(tempMessage), 0);
}
Sleep(1000);
}
return 0;
}

```

2.1.2 Receive 模块

Accept 线程为每一个 client 开辟了一个 Receive 线程，负责接收该 client 发送的消息并转发给对应的其他 client。

1. 线程首先根据线程参数确定对应的 client 结构体，接着开启 while 循环接收 client 发送的消息
2. 接收到消息后先查看是否是 exit 请求，如果是则需要关闭该套接字以及线程，并将 Client 结构体留出给后续使用
3. 若不是 exit 请求，则将接收到的消息放进对应的 Client 结构体的 buffer 中，然后开启一个转发线程独立完成发送工作

额外开一个转发线程是为了消息的接收和发送能够异步进行，不会因为发送而阻塞接收；同时将代码模块分离，使得程序更加规范简洁。

```

// 接受数据线程
unsigned __stdcall ThreadRecv(void* param){
    SOCKET client = INVALID_SOCKET;
    int index = 0;
    for (int j = 0; j < THREAD_NUM; j++)
        // 判断是为哪个客户端开辟的线程
        ...
    char temp[128] = { 0 }; // 临时数据缓冲区
    while (true)
    {
        memset(temp, 0, sizeof(temp));
        // 接收数据
        if (recv(client, temp, sizeof(temp), 0) == SOCKET_ERROR) // 忙等待
            continue;

        // 接收到数据，放进对应结构体的buffer
    }
}

```

```

memcpy(Clients[index].buffer, temp, sizeof(Clients[index].buffer));

if (strcmp(temp, "exit") == 0) // 如果客户发送exit请求, 那么直接关闭线程, 不打开转发线程
{
    // 告诉其他线程该用户下线
    ...
    closesocket(Clients[index].sClient); // 关闭该套接字
    CloseHandle(HandleRecv[index]); // 关闭线程句柄
    Clients[index].sClient = INVALID_SOCKET; // 位置空出来, 留给以后进入的客户使用
    Clients[index].flag = 0;
    HandleRecv[index] = NULL;
    // 打印系统提示
    ...
    break;
}
else
{ // 打印系统提示
    ...
    _beginthreadex(NULL, 0, ThreadSend, &index, 0, NULL); //
        开启一个转发线程, flag标记着这个需要被转发的信息是从哪个线程来的;
    }
}
return 0;
}

```

2.1.3 Send 模块

Send 线程是由 Receive 线程开启的, 负责将接收到并存放在对应结构体 buffer 中的消息转发给对应的 client。

1. 首先使用 `sprintf_s` 格式化 Client 结构体中的 buffer, 将发送源的用户名添加进消息中
2. 接着根据 Client 结构体中的 transID 字段判断消息的转发范围, 如果是 all 则向所有在线的 client 发送消息, 否则向指定的 client 发送消息

需要注意的是, 使用 `sprintf_s` 格式化消息时, 如果直接使用以下代码会出现错误:

```
sprintf_s(Clients[index].buffer, "[%s]: %s", Clients[index].userName, Clients[index].buffer)
```

因为 buffer 中的内容会先被替换一次, 下一个 %s 替换的将不再是原 buffer, 所以需要 temp 变量来进行存储。

另外, 在转发遍历 Client 结构体时, 除了让下标不等于当前线程, 即不给在自己发送消息以外, 还需要确认对应线程是否有效, 否则会导致消息发送错误而终止整个发送线程。

```

// 发送数据线程
unsigned __stdcall ThreadSend(void* param){
    int index = *(int*)param;
    //sprintf_s(Clients[index].buffer, "[%s]: %s", Clients[index].userName,
        Clients[index].buffer); // 逐步替换, 这样会错
}

```

```

char temp[128] = { 0 }; // 临时数据缓冲区, 存放接收到的数据
memcpy(temp, Clients[index].buffer, sizeof(temp));
sprintf_s(Clients[index].buffer, "[%s]: %s", Clients[index].userName, temp); //
    格式化: 把发送源的名字添加进信息里

if (strlen(temp) != 0) // 如果数据不为空且还没转发则转发
{
    if (strcmp(Clients[index].transID, "all") == 0) // 如果是给所有人转发
        for (int j = 0; j < THREAD_NUM; j++)
            // 需要确保当前client有效, 否则发送失败了后续都不发了
            if (j != index && Clients[j].sClient != INVALID_SOCKET) //
                向除自己之外的所有客户端发送信息
                if (send(Clients[j].sClient, Clients[index].buffer, sizeof(Clients[j].buffer), 0) ==
                    SOCKET_ERROR)
                    return 1;
    else
        for (int j = 0; j < THREAD_NUM; j++) // 向指定的人转发
            if (Clients[j].sClient != INVALID_SOCKET && j != index && strcmp(Clients[j].userName,
                Clients[index].transID) == 0) // 可以重名了
                if (send(Clients[j].sClient, Clients[index].buffer, sizeof(Clients[j].buffer), 0) ==
                    SOCKET_ERROR)
                    return 1;
}
return 0;
}

```

2.2 Client

2.2.1 Send 模块

client 发送端在 main 函数里使用一个 while 循环实现。

while 循环里首先打印时间和用户名信息; 然后使用 cin.getline() 函数接收用户输入的信息, 如果用户输入的是 exit, 则退出程序, 否则将用户输入的信息发送给 server 端。

```

// 接收用户输入并发送
char sendBuffer[128] = { 0 };
while (1)
{
    // 打印时间及用户名
    ...
    cin.getline(sendBuffer, 128);
    if (strcmp(sendBuffer, "exit") == 0)
    {
        cout << "you are going to exit... " << endl;
        Sleep(1000);
        if (send(ClientSocket, sendBuffer, sizeof(sendBuffer), 0) == SOCKET_ERROR)
            return -1;
        break;
    }
}

```

```

    }
    // 正常输入
    if (send(ClientSocket, sendBuffer, sizeof(sendBuffer), 0) == SOCKET_ERROR)
        return -1;
}

```

2.2.2 Receive 模块

前面提到，client 端发送了聊天对象后，会开辟一个线程负责接收消息，该线程内使用 while 循环不断阻塞接收。

接收到消息后不能直接打印，需要先覆盖掉当前 client 的用户名再进行打印：使用 '\b' 实现退格操作，覆盖掉之前打印的用户名；打印完接收到的消息后再重新输出被覆盖的用户名。

```

// 接收数据线程
unsigned __stdcall ThreadRecv(void* param)
{
    char bufferRecv[128] = { 0 };
    while (true)
    {
        if (recv(*(SOCKET*)param, bufferRecv, sizeof(bufferRecv), 0) == SOCKET_ERROR)
        {
            Sleep(500);
            continue;
        }
        if (strlen(bufferRecv) != 0)
        {
            // 覆盖之前的用户名
            for (int i = 0; i <= strlen(userName) + 26; i++)
                cout << "\b";
            // 打印时间与接收到的信息
            ...
            cout << bufferRecv << endl;
            // 重新打印userName，因为之前覆盖掉了
            time(&now_time); // 打印时间
            strftime(now_time_str, 20, "%Y-%m-%d %H:%M:%S", localtime(&now_time));
            cout << "(" << now_time_str << ") ";
            cout << "[" << userName << "]: "; // 打印用户名
        }
        else
            Sleep(100);
    }
    return 0;
}

```


3 运行展示

3.1 程序启动

图3.2是启动服务器和客户端以后的连接过程，左上角是服务器，另外三个是客户端。

- 可以看到用户 t1, t2 已经连接，对话界面已经刷新为聊天页面，在服务器端有连接成功的消息提示，t1 也能看到 t2 上线的提示。
- 此时 t3 正在与服务器建立连接，服务器向 t3 发送了当前聊天室中的其他用户，t3 输入用户名，选择聊天对象。

```

D:\BFF\src\server>
*****Chat Room*****
=====
please wait for other clients to join...
if you want to close the server, please input 'exit'
=====
2023-10-18 18:56:32
[system]: 't1' connect successfully!
2023-10-18 18:56:52
[system]: 't2' connect successfully!

D:\BFF\src\client>
*****Chat Room*****
=====
(2023-10-18 18:56:52) [system]: t2 come!
(2023-10-18 18:56:52) [t1]: |

D:\BFF\src\client>
*****Chat Room*****
=====
connecting...please wait...
Connect successfully!
The server's ip is:127.0.0.1 The server's port is:1818
2023-10-18 18:56:56
=====
At present these people are in the chat room:
[t1]
[t2]
=====
Please input your name: t3
Hi t3!
=====
You can choose other clients to chat:
- If you input 'all', then everyone can receive your message
- If you input the client's name, then the only client can receive your
age
please choose the client:

```

图 3.2: 程序连接

3.2 消息发送

图3.3是程序运行过程中互相发送消息的展示。t1、t2 选择的发送对象为 ‘all’，t3 选择的对象为 t2。

- t3 发送消息，只有 t2 能够接收到对应的消息，服务器端显示的记录也只是发送给 t2。
- t2 向所有人发送了“我是 t2”的信息，从图中可以看出经过一定延时后，服务器将该信息转发给了另外两名用户，并且在服务器上留下了 “[t2]: 我是 t2 (to 'all')”的记录。

```

*****Chat Room*****
=====
please wait for other clients to join...
if you want to close the server, please input 'exit'
=====
2023-10-18 18:56:33
[system]: 't1' connect successfully!
2023-10-18 18:56:52
[system]: 't2' connect successfully!
2023-10-18 18:57:22
[system]: 't3' connect successfully!
2023-10-18 18:57:27
[t3]: hi (to 't2')
2023-10-18 18:57:35
[t2]: 我是t2 (to 'all')
2023-10-18 18:57:39
[t1]: ok (to 'all')
|

*****Chat Room*****
=====
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:27) [t3]: hi
(2023-10-18 18:57:27) [t2]: 我是t2
(2023-10-18 18:57:39) [t1]: ok
(2023-10-18 18:57:39) [t2]: |

D:\BFF\src\client>
*****Chat Room*****
=====
(2023-10-18 18:56:52) [system]: t2 come!
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:35) [t2]: 我是t2
(2023-10-18 18:57:35) [t1]: ok
(2023-10-18 18:57:39) [t1]:

D:\BFF\src\client>
*****Chat Room*****
=====
(2023-10-18 18:57:22) [t3]: hi
(2023-10-18 18:57:25) [t2]: 我是t2
(2023-10-18 18:57:39) [t1]: ok
(2023-10-18 18:57:39) [t3]: |

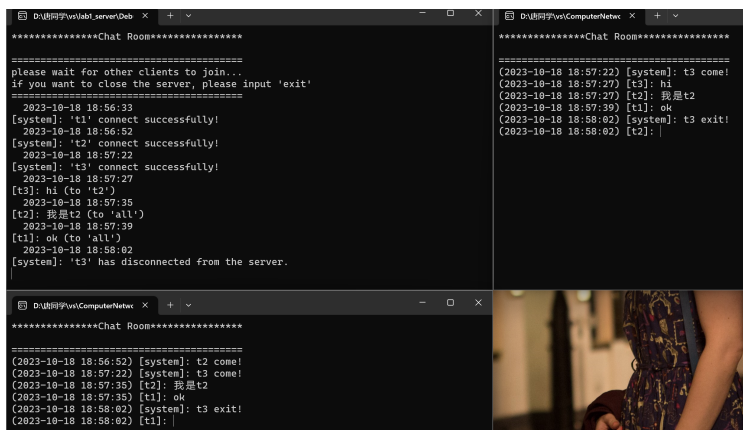
```

图 3.3: 消息发送

3.3 程序退出

t3 输入 exit，与服务器端断开连接，退出程序。

由下图可以看到，t3 程序退出，服务器显示 t3 disconnected 记录，另外两名用户也收到了 t3 下线的提示。



```
*****Chat Room*****
=====
please wait for other clients to join...
if you want to close the server, please input 'exit'
=====
2023-10-18 18:56:33
[system]: 't1' connect successfully!
2023-10-18 18:56:52
[system]: 't2' connect successfully!
2023-10-18 18:57:22
[system]: 't3' connect successfully!
2023-10-18 18:57:27
[t3]: hi (to 't2')
2023-10-18 18:57:39
[t2]: 我是t2 (to 'all')
2023-10-18 18:57:39
[t1]: ok (to 'all')
2023-10-18 18:58:02
[system]: 't3' has disconnected from the server.

*****Chat Room*****
=====
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:27) [t3]: hi
(2023-10-18 18:57:27) [t2]: 我是t2
(2023-10-18 18:57:39) [t1]: ok
(2023-10-18 18:58:02) [system]: t3 exit!
(2023-10-18 18:58:02) [t2]: |

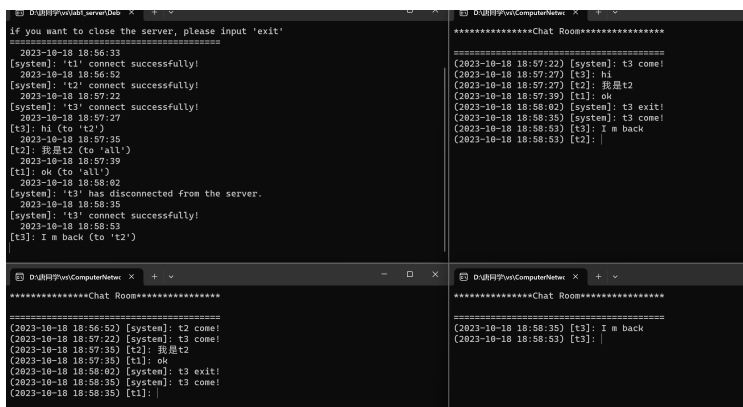
*****Chat Room*****
=====
(2023-10-18 18:56:52) [system]: t2 come!
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:35) [t2]: 我是t2
(2023-10-18 18:57:35) [t1]: ok
(2023-10-18 18:58:02) [system]: t3 exit!
(2023-10-18 18:58:02) [t1]: |
```

图 3.4: 程序退出

3.4 再次上线

接着再打开一个客户端，让 t3 重新进入聊天室。

从图3.5可以看到，当 t3 重新申请连接服务器时，服务器返回当前聊天室只有 t1、t2 两人，证明服务器已经了解之前 t3 下线情况。t3 再次向 t2 发送信息，可以看出服务器正确响应，t2 正常接收。



```
*****Chat Room*****
=====
if you want to close the server, please input 'exit'
=====
2023-10-18 18:56:33
[system]: 't1' connect successfully!
2023-10-18 18:56:52
[system]: 't2' connect successfully!
2023-10-18 18:57:22
[system]: 't3' connect successfully!
2023-10-18 18:57:27
[t3]: hi (to 't2')
2023-10-18 18:57:35
[t2]: 我是t2 (to 'all')
2023-10-18 18:57:39
[t1]: ok (to 'all')
2023-10-18 18:58:02
[system]: 't3' has disconnected from the server.
2023-10-18 18:58:35
[system]: 't3' connect successfully!
2023-10-18 18:58:53
[t3]: I m back (to 't2')

*****Chat Room*****
=====
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:27) [t3]: hi
(2023-10-18 18:57:27) [t2]: 我是t2
(2023-10-18 18:57:39) [t1]: ok
(2023-10-18 18:58:02) [system]: t3 exit!
(2023-10-18 18:58:35) [system]: t3 come!
(2023-10-18 18:58:53) [t3]: I m back
(2023-10-18 18:58:53) [t2]: |

*****Chat Room*****
=====
(2023-10-18 18:56:52) [system]: t2 come!
(2023-10-18 18:57:22) [system]: t3 come!
(2023-10-18 18:57:35) [t2]: 我是t2
(2023-10-18 18:57:35) [t1]: ok
(2023-10-18 18:58:02) [system]: t3 exit!
(2023-10-18 18:58:35) [system]: t3 come!
(2023-10-18 18:58:35) [t1]: |

*****Chat Room*****
=====
(2023-10-18 18:58:35) [t3]: I m back
(2023-10-18 18:58:53) [t3]: |
```

图 3.5: 重新上线

3.5 日志记录

除了在控制台输出消息记录外，Server 还会向日志文件中输出以保存信息，图3.6展示了程序运行的日志记录。

可以看到之前在控制台的记录信息都保存到了日志文件中，最后每个客户端断开连接后，服务器端输入 eixt 退出程序，提示信息“The server is closed.”也被记录在了日志文件中。



```
output.log
文件 编辑 查看

*****Chat Room*****

=====
2023-10-18 18:56:33
[system]: 't1' connect successfully!
2023-10-18 18:56:52
[system]: 't2' connect successfully!
2023-10-18 18:57:22
[system]: 't3' connect successfully!
2023-10-18 18:57:27
[t3]: hi (to 't2')
2023-10-18 18:57:35
[t2]: 我是t2 (to 'all')
2023-10-18 18:57:39
[t1]: ok (to 'all')
2023-10-18 18:58:02
[system]: 't3' has disconnected from the server.
2023-10-18 18:58:35
[system]: 't3' connect successfully!
2023-10-18 18:58:53
[t3]: I m back (to 't2')
2023-10-18 18:59:08
[system]: 't3' has disconnected from the server.
2023-10-18 18:59:10
[system]: 't2' has disconnected from the server.
2023-10-18 18:59:12
[system]: 't1' has disconnected from the server.
The server is closed.
```

图 3.6: 日志记录

4 实验总结

4.1 实验遇到的问题

1. 实验遇到的第一个问题是对 Socket 函数的使用不熟悉，对于接口的调用不清楚。在查阅相关资料并且阅读一些参考代码后，逐渐对需要使用的函数等有了一定的认识，对程序的实现有了一个大致的框架。
2. 接着是对多线程的使用与维护，虽然上学期学习了并行程序设计的课程，但经过一段时间的松懈，才发现对于该方面的知识掌握并不牢固。
3. 具体的代码实现上，像在2.1.3节中提到的，sprintf 函数的使用有个不小的坑；在消息发送时需要检查 Client 结构体排除无效情况；另外各个客户端（线程）的维护也是个不小的麻烦...

4.2 实验思考

本次实验学习了使用 Socket 编程聊天程序，掌握了 Socket 基本使用方法，巩固了多线程的使用以及计算机网络理论课上所学的知识。

另外，本次实验做的程序与协议设计上仍然有改进空间：比如让用户选择是否接收群发消息；让用户查询当前聊天室内的用户；让用户改变聊天对象等。