

计算机网络第一次实验

——利用Socket，编写一个聊天程序

1811507 文静静

- 实验要求
- 实验内容
 - 聊天协议：
 - 数据包
 - 客户端与服务端时序
 - 实现思路：
 - 服务器处理客户端连接请求
 - 服务器管理客户端连接
 - 客户端发送消息
 - 服务端接收消息
 - 服务端转发消息
 - 客户端接收消息
 - 客户端退出聊天
- 功能展示
 - 服务器开启
 - 客户端开启
 - 多客户端进入聊天室
 - 客户端聊天
 - 客户端群聊
 - 客户端私聊
 - 客户端退出聊天

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。不允许使用CSocket等封装后的类编写程序。(3) 使用流式Socket完成程序。(4) 程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。(5) 完成的程序至少应能实现两个用户之间的英文和中文聊天。(6) 编写的程序应结构清晰，具有较好的可读性。

实验内容

由于两个用户之间聊天相当于多人聊天中的特殊情况，所以之间实现多人聊天，既可在聊天室中多人聊天（即服务器将消息转发给所有客户端），也可与某客户端单独聊天（即服务器只将消息转发给特定客户端，若找不到该客户端，将反馈给发送者）。

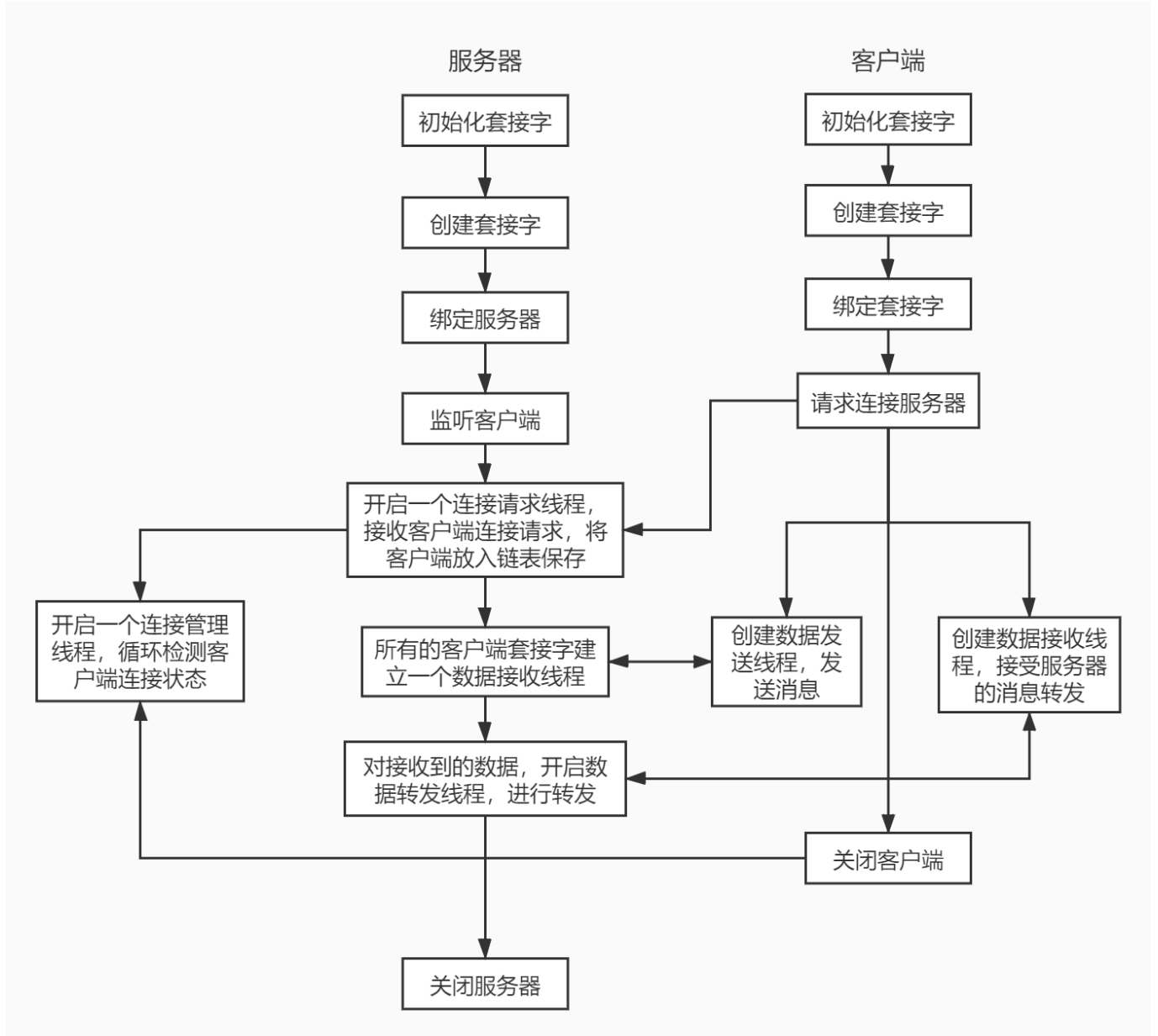
聊天协议：

数据包

- 客户端->服务端
 - 一个字段，由中文、英文和字符组成，表示客户端用户名；

- 一个字段，由中文、英文和字符组成，表示客户端发送的消息；
- 两个字段，第一个字段为#，标识后面为消息接受方的用户名，第二个字段为接收方用户名(中文、英文和字符串)组成。
- **服务端->客户端**
 - 由三个字段组成，第一个字段显示是群聊还是私聊，第二个字段显示发送者用户名，第三个字段为消息内容。

客户端与服务端时序



- 当客户端请求连接，若此时客户端连接数未达到最大值，且客户端位于等待队列队首时，客户端与服务端进行连接；
- 当客户端与服务端进行连接之后，客户端与客户端之间的消息交互由服务器进行转发完成，先发送消息接收方用户名，再发送消息，服务器根据接受发用户名进行转发。
- 当客户端选择退出聊天之后，关闭套接字与连接，服务器连接状态线程检测到中断连接即进行处理。

实现思路：

使用链表和结构实现客户端保存

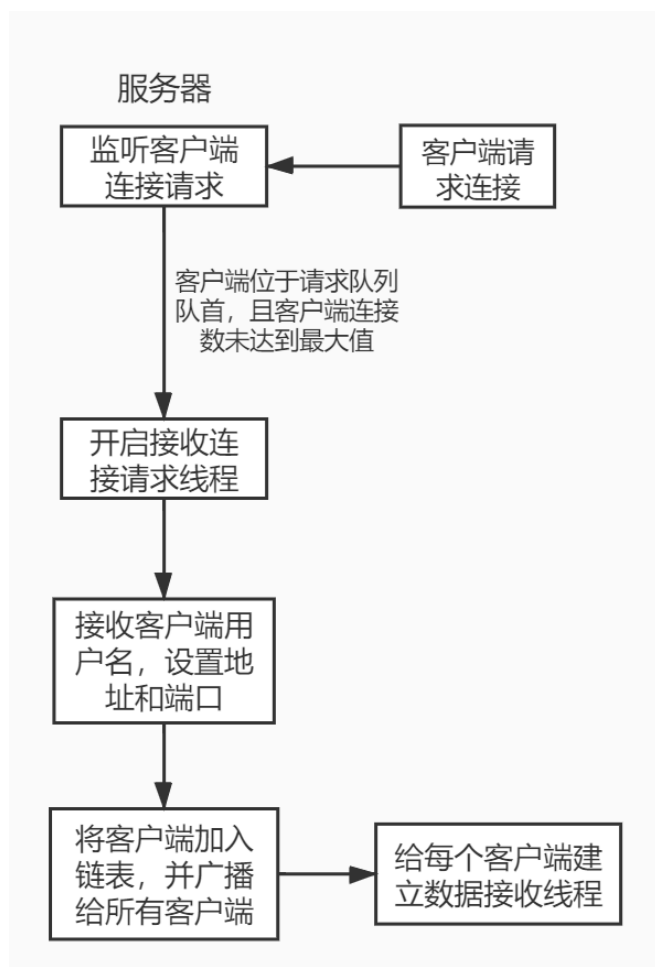
```

// 客户端结构
typedef struct client {
    SOCKET clients; // 客户端套接字
    char buff[255]; // 数据缓冲区
    char username[31]; // 客户端用户名
    char chatname[31]; // 聊天对象用户名
    char ip[20]; // 客户端IP
    unsigned short port; // 客户端端口
    UINT_PTR flag; // 标记客户端，用来区分不同客户端
    client* next;
}*pclient;

//初始化链表
void init();
//获取头结点
pclient getheadnode();
//添加一个客户端
void addclient(pclient c);
//删除一个客户端
bool deleteclient(UINT_PTR flag);
//发送消息
void senddata(pclient c);
//检查连接状态
void checkconnection();
//清空链表
void cleanclient();

```

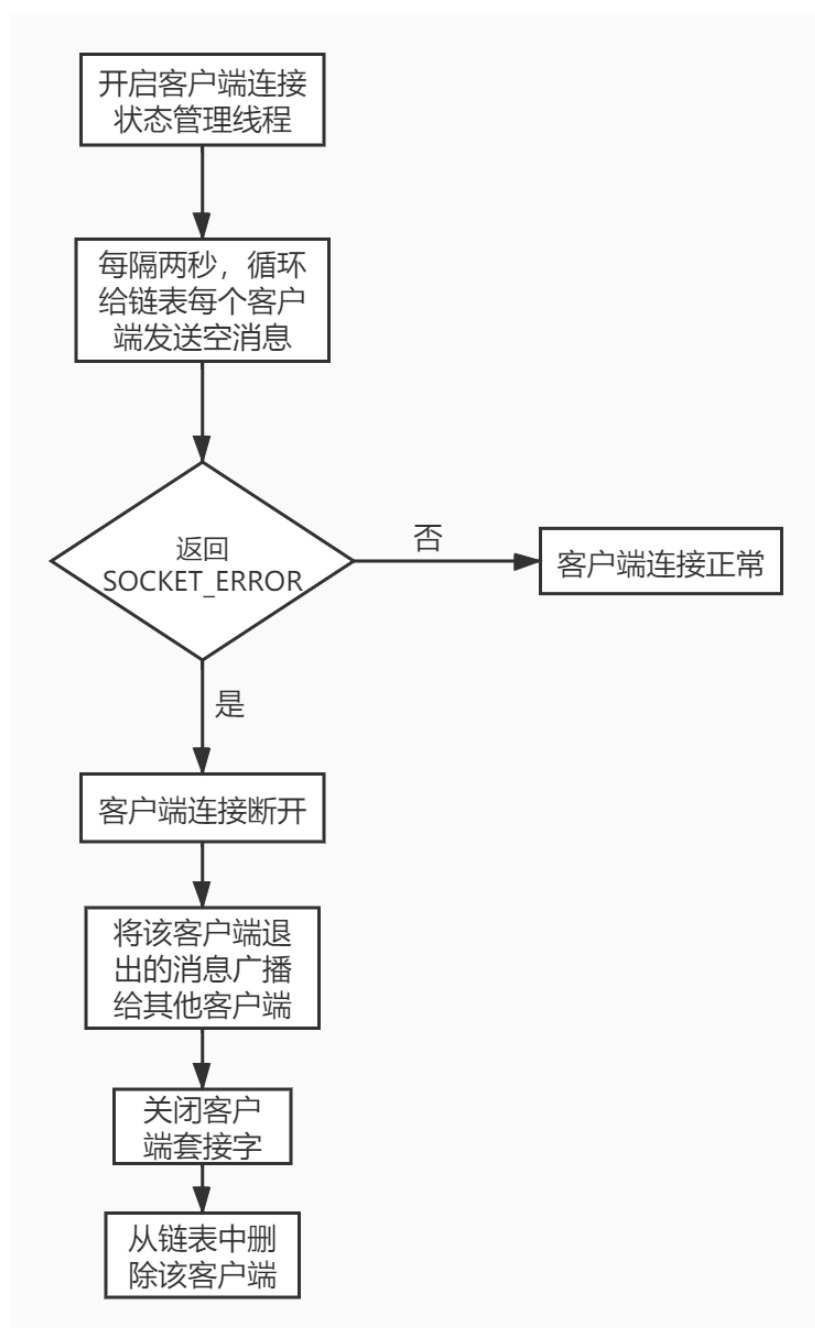
服务器处理客户端连接请求



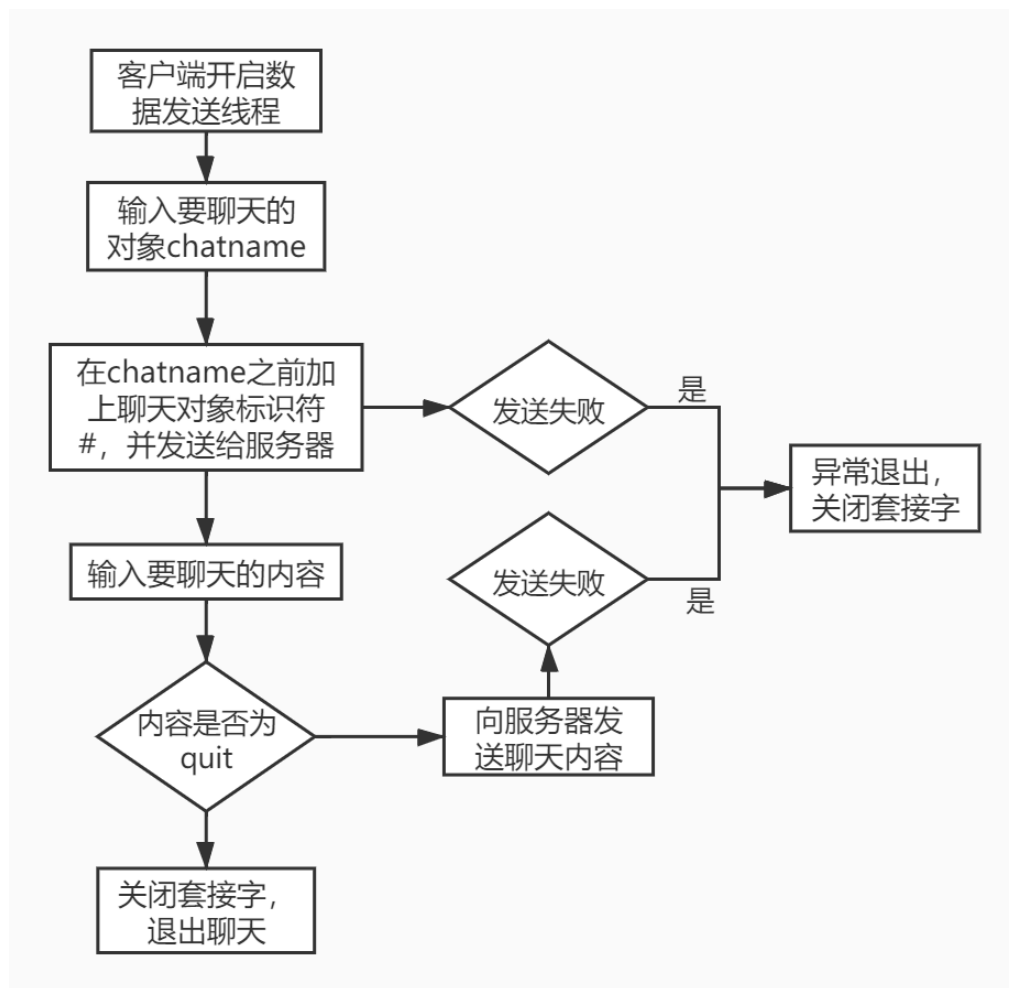
其中将客户端加入链表并通过遍历链表，将该客户端加入聊天的消息广播给所有客户端：

```
void addclient(pclient c) {  
    // 将客户端加入链表  
    c->next = head->next;  
    head->next = c;  
    pclient p = getheadnode();  
    strcpy(c->buff, c->username);  
    strcat(c->buff, "加入聊天");  
    // 向链表中所有的客户端广播该客户端加入聊天  
    while (p = p->next) {  
        if (p->flag != c->flag) {  
            send(p->clients, c->buff, sizeof(c->buff), 0);  
        }  
    }  
}
```

服务器管理客户端连接



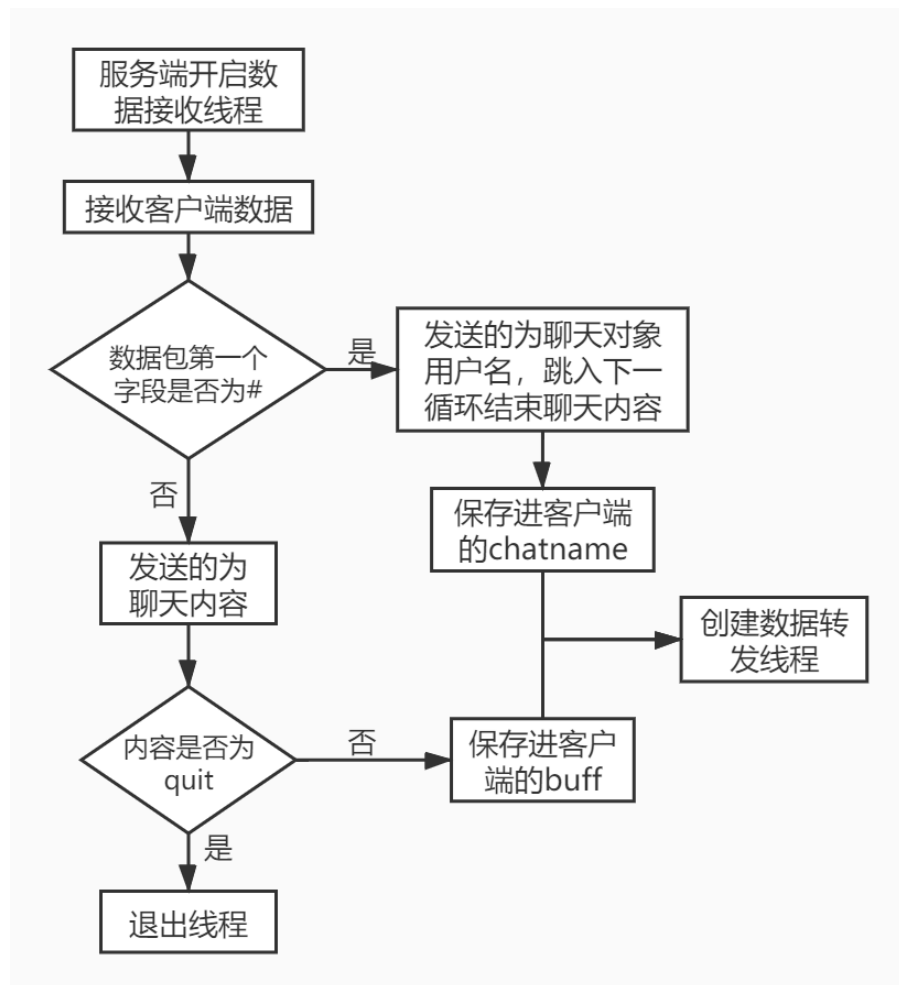
客户端发送消息



一定要先发送聊天对象用户名，再发送聊天内容。

```
DWORD WINAPI SendThread(LPVOID lpParameter) {
    char buff[255] = { 0 };
    int res1 = 0, res2 = 0;
    while (1) {
        memset(chatname, 0, sizeof(chatname));
        memset(buff, 0, sizeof(buff));
        // 输入要聊天的对象
        cin.getline(chatname, 32);
        // 在聊天对象用户名面前加标识符#并发送
        strcpy(buff, "#");
        strcat(buff, chatname);
        res1 = send(*(SOCKET*)lpParameter, buff, sizeof(buff), 0);
        if (res1 == SOCKET_ERROR) {
            cout << "发送消息失败!" << endl << endl;
            return -1;
        }
        memset(buff, 0, sizeof(buff));
        // 输入要聊天的内容
        cin.getline(buff, 255);
        cout << endl;
        // 如果输入结束信号"quit", 那么将结束信号发送给服务器并关闭套接字, 显示退出聊天。
        if (strcmp(buff, "quit") == 0) {
            res2 = send(*(SOCKET*)lpParameter, buff, sizeof(buff), 0);
            if (res2 == SOCKET_ERROR) {
                cout << "发送消息失败!" << endl << endl;
                return -1;
            }
            closesocket(ClientSocket);
            cout << "已退出聊天。" << endl << endl;
            WSACleanup();
            return 0;
        }
        // 将输入的消息发送给服务端
        res2 = send(*(SOCKET*)lpParameter, buff, sizeof(buff), 0);
        if (res2 == SOCKET_ERROR) {
            cout << "发送消息失败!" << endl << endl;
            return -1;
        }
    }
}
return 0;
}
```

服务端接收消息

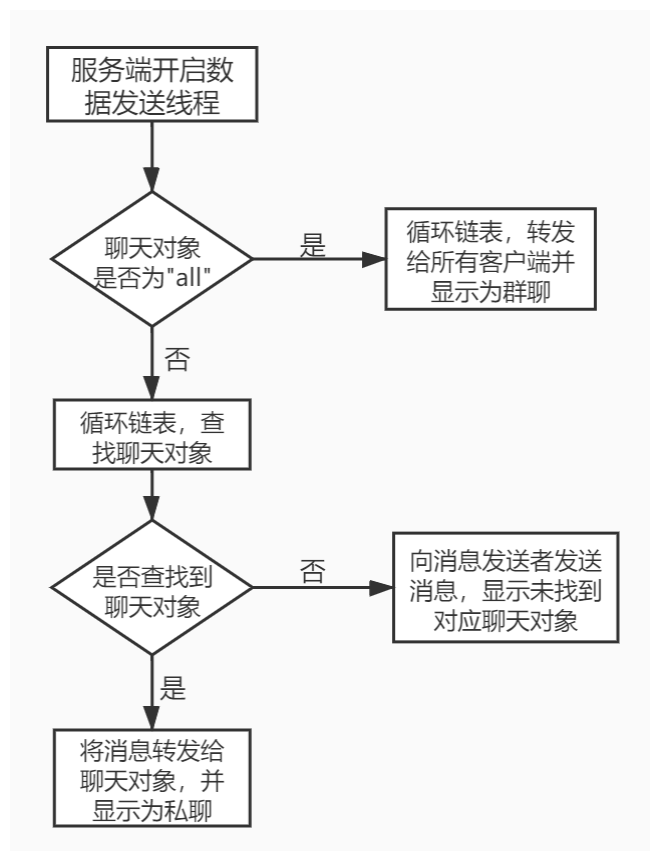


```

//接受数据
DWORD WINAPI RevThread(LPVOID lpParameter) {
    int res = 0;
    char chatname[31];
    char buff[255];
    while (1) {
        pclient p = (pclient)lpParameter;
        res = recv(p->clients, buff, sizeof(buff), 0);
        if (res == SOCKET_ERROR) {
            return -1;
        }
        // 如果第一个字符为#, 则代表发送的不是消息而是聊天对象
        if (buff[0] == '#') {
            int i = 1;
            // 将聊天对象保存, 然后开始下一轮循环接收消息
            while (buff[i] != '\0') {
                p->chatname[i - 1] = buff[i];
                i++;
            }
            p->chatname[i - 1] = '\0';
            continue;
        }
        // 如果是客户端发来的退出信号"quit", 就退出线程
        // 因为客户端以及退出关闭套接字, 所以服务器的检测连接状态的线程会检测到并广播给所有客户端
        if (strcmp(buff, "quit") == 0) {
            return 0;
        }
        else {
            // 保存聊天内容
            strcpy(p->buff, buff);
            CreateThread(NULL, 0, &SendThread, p, 0, NULL);
        }
    }
    return 0;
}

```

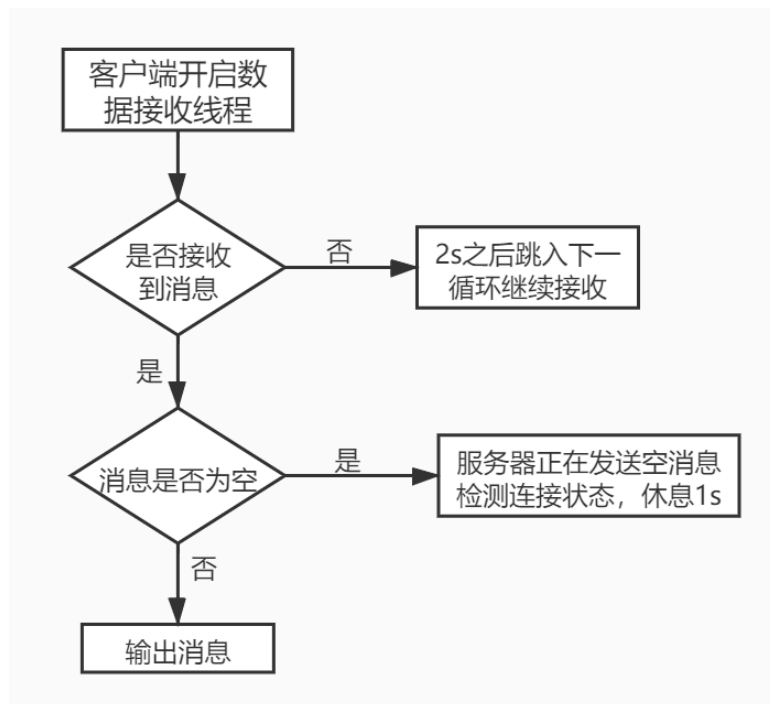

服务端转发消息



```

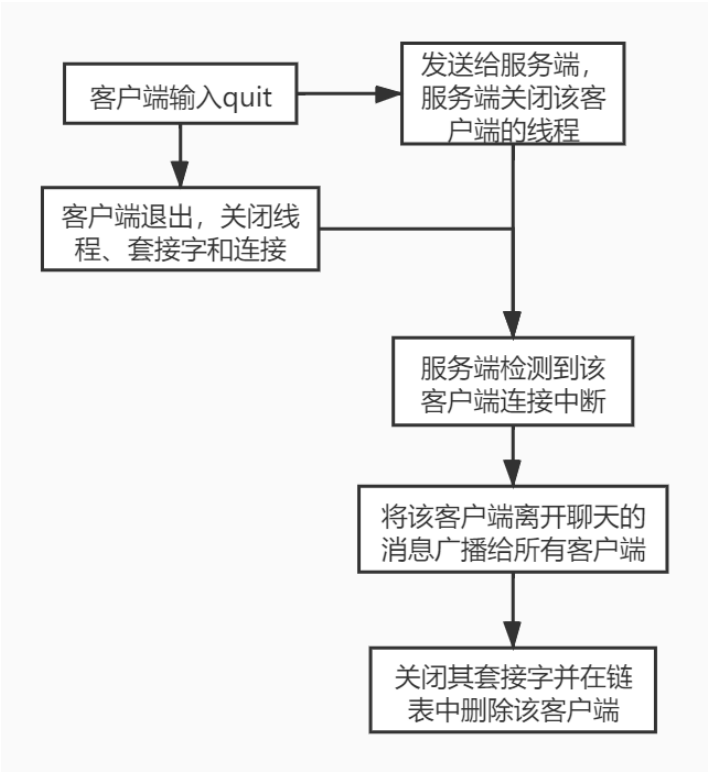
void senddata(pclient c) {
    // 如果接收人是all，那么将消息广播给所有客户端，并显示为群聊
    if (strcmp(c->chatname, "all") == 0) {
        cout << "(群聊)" << c->username << ": " << c->buff << endl << endl;
        pclient p = getheadnode();
        while (p = p->next) {
            if (p->flag != c->flag) {
                // 将消息转变成 (群聊)username: text的形式
                strcpy(p->buff, "(群聊)");
                strcat(p->buff, c->username);
                strcat(p->buff, ": ");
                strcat(p->buff, c->buff);
                send(p->clients, p->buff, sizeof(p->buff), 0);
            }
        }
    }
    else {
        pclient p = getheadnode();
        // 如果接收者是某个客户端，只将消息转发给该客户端
        while (p = p->next) {
            if (p->flag != c->flag && strcmp(p->username, c->chatname) == 0) {
                // 将消息转变成 (私聊)username: text的形式
                strcpy(p->buff, "(私聊)");
                strcat(p->buff, c->username);
                strcat(p->buff, ": ");
                strcat(p->buff, c->buff);
                send(p->clients, p->buff, sizeof(p->buff), 0);
                cout << "(私聊 " << c->username << " to " << c->chatname << " ): " << c->buff << endl << endl;
                return;
            }
        }
        // 如果没有找到要转发的对象，将该消息回复给发送者
        strcpy(c->buff, "您要聊天的对象不在线上。");
        send(c->clients, c->buff, sizeof(c->buff), 0);
    }
}
  
```

客户端接收消息



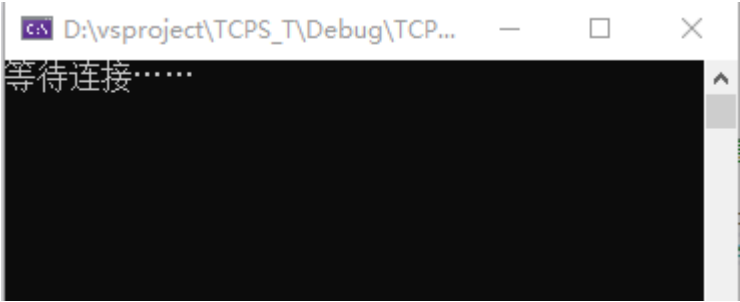
```
DWORD WINAPI RevThread(LPVOID lpParameter) {
    char buff[255] = { 0 };
    int res = 0;
    while (1) {
        res = recv(*(SOCKET*)lpParameter, buff, sizeof(buff), 0);
        // 如果没有收到消息, 就2s之后再次接收消息
        if (res == SOCKET_ERROR) {
            Sleep(2000);
            continue;
        }
        // 如果接收到非空消息, 就输出消息
        if (strlen(buff) != 0) {
            cout << buff << endl << endl;
        }
        else
            // 接收到空消息 (服务器在通过发送空消息测试连接状态)
            Sleep(100);
    }
    return 0;
}
```

客户端退出聊天



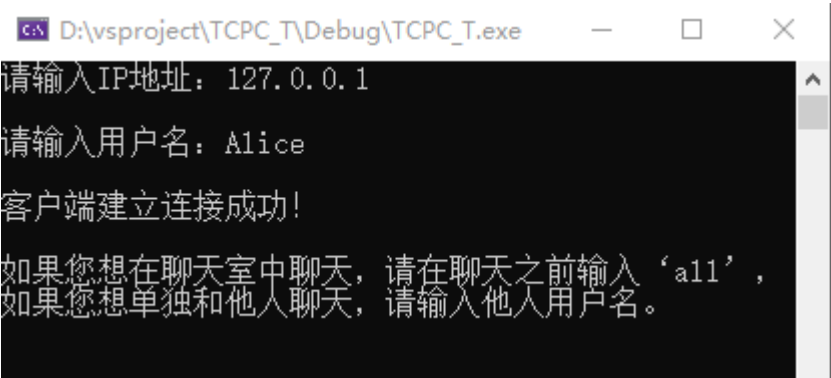
功能展示

服务器开启

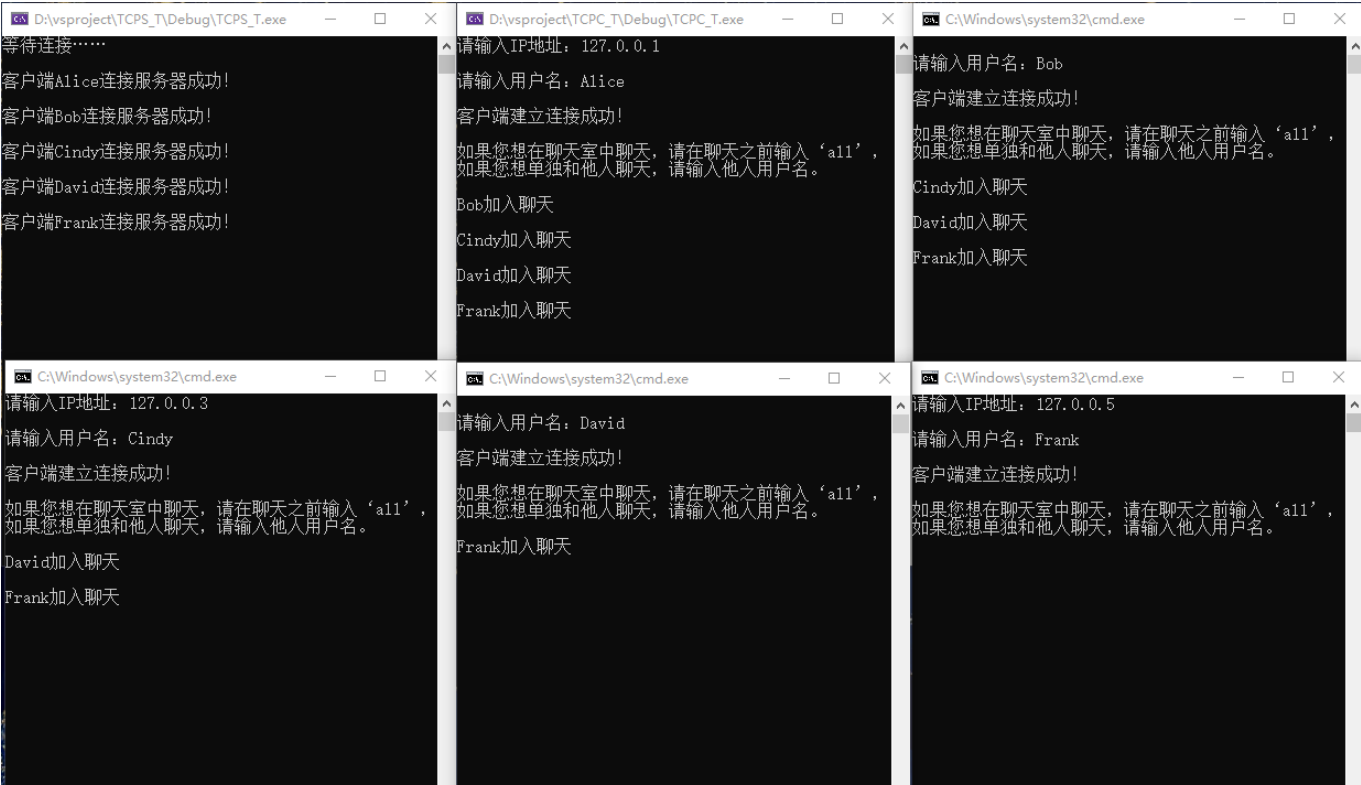


客户端开启

输入IP地址和用户名



多客户端进入聊天室



客户端聊天

可中英文自由聊天，且服务器界面会保存所有聊天，无论群聊还是私聊。（为了使聊天界面看起来清晰，多输出了一个换行）

客户端群聊

在发送聊天信息之前，输入all即表示进行群聊



客户端私聊

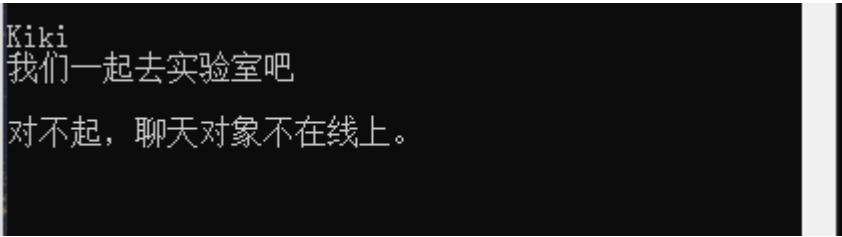
在发送聊天信息之前，输入消息接收者的用户名表示与其进行私聊

当聊天对象存在时：



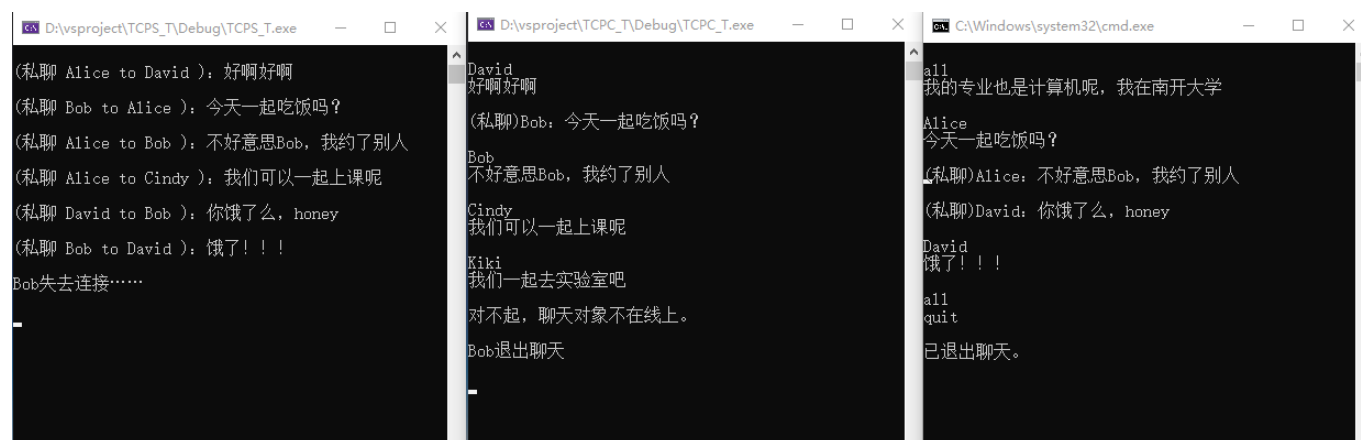
当聊天对象不存在时：

服务器如果找不到客户端传入的聊天对象，那么反馈给客户端，告诉它找不到聊天对象。



客户端退出

当客户端输入quit主动退出或者关闭客户端程序时，服务器将会检测到连接断开，并将该客户端离开群聊的消息广播给所有客户端，然后关闭该客户端套接字和线程，在链表中删除客户端，显示其失去连接。



The image displays three side-by-side screenshots of a chat application's user interface, showing a sequence of events where a client (Bob) disconnects. The windows are titled 'D:\vsproject\TCPS_T\Debug\TCPS_T.exe' and 'C:\Windows\system32\cmd.exe'.

Left Screenshot (Client View): Shows a chat log with messages from Alice, Bob, and David. The last message is 'Bob失去连接……' (Bob lost connection...). The input field is empty.

Middle Screenshot (Server/Client View): Shows the chat log from the server's perspective. It lists the names of all connected clients: David, Alice, Bob, Cindy, and Kiki. The last message is 'Bob退出聊天' (Bob exited chat).

Right Screenshot (Command Prompt View): Shows the command prompt output. It displays the 'quit' command being entered, followed by the message '已退出聊天。' (Already exited chat.).