实验一　使用 Socket 编写聊天程序

一、实验介绍

本实验要使用 Socket 编写一个聊天程序，要求如下：

1.给出聊天协议的完整说明。

2.使用 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，使用流式 Socket 完成程序。

3.程序应有基本对话界面、正常退出方式。

4.至少实现两个用户之间的中英文聊天，尝试扩展其他功能。

二、实验思路

1.整体框架

使用 winsock 网络编程接口，基于 TCP 协议，分为服务端和客户端分别进行实现。使用 socket，bind，listen，accept，connect，recv，send 等基本函数。注意根据返回值判断目的操作是否成功。

在服务器端和客户端分别使用多线程进行消息通信，使用 CreateThread 函数。

使用 char 或 string 类型的字符串处理消息内容，一些情况需要转换格式。消息内容默认可以支持中英文及任意语言，使用 getline()函数支持包含空格的字符串。

2.聊天协议说明

（1）客户端发送的消息格式

对于当前在线用户，支持用户间的私聊和面向所有用户的群聊。所以客户端发送的消息格式如下，在一条字符串中用“@”分隔消息内容和发送对象：

私聊：消息内容+@+要私聊的用户名，如 message@user_name

群聊：消息内容+@+all/消息内容+@+所有人，如 message@all，message@所有人

服务器在接收到这样格式的用户消息后，解析时将“@”替换为“#”，将接收端标识改为发送端标识。再转发给接收端。

（2）服务器发送和客户端接收的消息格式

所以服务器发送和客户端接收的消息格式为：message#send_name，客户端将此消息解析便能得到发送端信息和消息内容。

3.功能实现

（1)程序使用命令行界面，适当修饰，有正常的退出方式：客户端输入“exit”退出窗口。

（2）程序实现了在线用户的任意语言的聊天。

（3）实现了一对一私聊和所有在线用户的群聊。

尚不完全支持自定义创建的群聊，但已经实现了服务端识别用户创建群聊的请求。（通过解析用户的一条消息实现），后续可构造 map 结构存储并处理。

（4）实现了服务器缓存消息，给未在线用户也能发送消息，用户上线后即接收到消息。

（5）实现了服务器存储用户信息列表，存储消息记录并输出到文件。

其他功能之后逐步完善。

三、代码分析

1.服务端 server.cpp 文件：

定义变量和函数：

```
#define C_NUM 20//能连接的客户端数量
int c_num = 0;//当前客户端数量
map<string, SOCKET> client_info;//存储用户信息
map<SOCKET, string> client_info2;//存储用户信息，存两次方便查找
void initialization();//初始化套接字库
void communicate(SOCKET s_accept);//服务器收发消息函数，线程中使用
map<string, string> msg_for_absent;//缓存给未在线用户发送的信息
```

主函数：

首先定义服务端套接字，接受请求套接字，服务端地址，客户端地址。接受请求套接字定义为 SOCKET 类型的数组，用来实现服务器与多个客户端的通信。

```
SOCKET s_server;
SOCKET s_accept[C_NUM] = { 0 };
SOCKADDR_IN server_addr;
SOCKADDR_IN client_addr;
```

初始化套接字库，检测版本号等，封装在 initialization()函数中。

```
WORD w_req = MAKEWORD(2, 2);//版本号
WSADATA wsadata;
WSAStartup(w_req, &wsadata);//成功返回 0
```

填充服务端信息，定义套接字，并使用 bind()函数绑定套接字和端口号，根据返回值判断是否绑定成功。

```
server_addr.sin_family = AF_INET;
server_addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(1234);
s_server = socket(AF_INET, SOCK_STREAM, 0);
if (bind(s_server, (SOCKADDR*)&server_addr, sizeof(SOCKADDR)) == SOCKET_ERROR) …
else…
```

使用 listen()函数设置套接字为监听状态，同样根据返回值检查。

```
if (listen(s_server, SOMAXCONN) < 0)
    …
else
    …
```

接下来进入 while 循环。

在每次循环中，当接收到一个客户端的连接时，提示用户注册用户名，在 map<string, SOCKET> client_info 数据结构中存储“用户名，accept()函数的 SOCKET 类型返回值”键值对，为了方便键值之间的互相查询还定义了一个键和值顺序相反的 map<string, SOCKET>client_info2。服务端还能显示当前连接上的用户列表信息（map 中为提高查找效率是按照特定顺序而不是插入顺序存储，所以显示的用户列表也不一定是按创建顺序的）。

这里还增加了检查用户名是否在 msg_for_absent 的键值中，如果是，说明此用户不在线是有人给其发消息，那么服务器现在就给他发送这条缓存的消息。

之后使用 CreateThread()函数开启一个针对此客户端的线程，函数为 communicate()，用于与客户端的通信（接收客户端的消息，解析，并转发给另一个或多个客户端）。

```
s_accept[c_num] = accept(s_server, (SOCKADDR*)&client_addr, &len);
client_info[recv_buf] = s_accept[c_num];
client_info2[s_accept[c_num]] = recv_buf;
for (map<string, SOCKET>::iterator iter = client_info.begin(); iter != client_info.end(); iter++)
        cout << iter->first << "    " << iter->second << endl;
CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)communicate, (LPVOID)s_accept[c_num], NULL, NULL);
```

communicate()函数内部主体是在一个无限 while 循环中，使用 recv()函数收取客户端信息并解析转发。

根据协议，用户发的消息有私聊和群聊两种类型，根据“@”符号分离其前后的内容得到消息主体和接收端用户名，根据 recv()函数的第一个参数 s_accept 可以获取发送端的 SOCKET 类型值，进而从 map 中得到发送端用户名。

服务器将以上两部分以“#”相连成一个字符串发送给接收端，接收端接收和解析后就能知道消息来源和内容了。

当接收端为所有人时，服务器遍历用户列表，将消息发给所有人。

对于发送消息给不在线用户，服务器也进行缓存，存储到 msg_for_absent 里。

```
//解析接收到的消息，如 A 发的：hello@B  改写为 hello#A(A 发的)，并发送给 B
string the_name = recv_buf.substr(i + 1, recv_buf.length() - 1);
string the_msg = recv_buf.substr(0, i);
string msg_processed = the_msg + '#' + client_info2[s_accept];
```

char send_msg[100]; strcpy(send_msg, msg_processed.c_str());

服务器还会在窗口显示用户间的聊天信息，将聊天记录输出到文件。
每次 while 循环后 c_num 自增 1。
最后关闭套接字，释放资源。

2.客户端 client.cpp 文件：

定义函数：

void initialization();//初始化套接字库
void receive(SOCKET s_server);//接收消息函数，在线程中使用

主函数：

前面类似服务端，定义服务端套接字，服务端地址，填充服务端信息，创建套接字。

之后使用 connect()函数通过三次握手来建立与 TCP 服务器的连接。根据返回值判断成功与否。连接成功后创建用户名作为标识，服务器端会在 map 中存储用户信息列表。接下来便可开始聊天。

用户可以输入#groupname,user_name1,user_name2…来创建更个性化的群聊。

用户可以输入 exit 退出，结束聊天。

首先进入 receive()函数的线程，主体内容也在无限 while 循环中，保证一直接收信息。接收服务器端的消息并解析，得到发送端信息和消息内容。输出到窗口。

再进入发送消息的循环中，这里使用 cin.getline()函数能支持包含空格的消息。

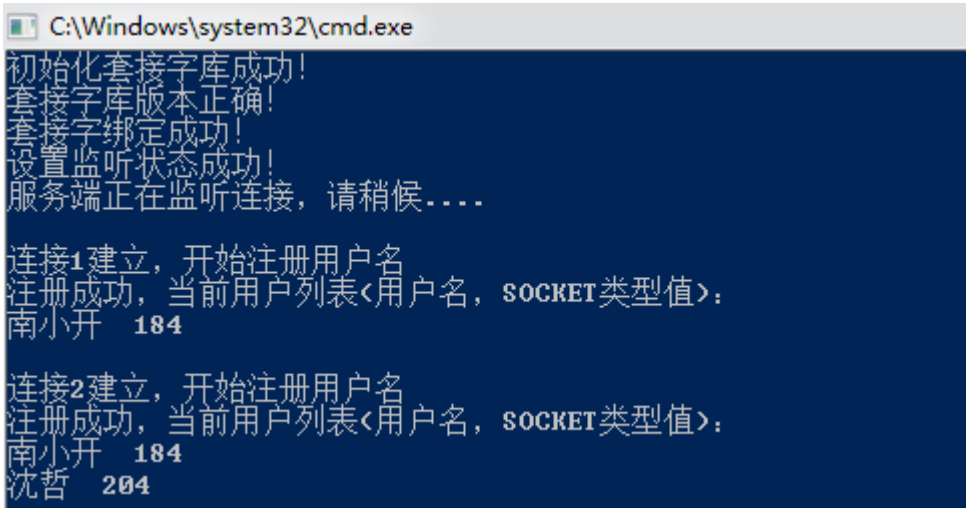
用户发送消息和接收消息是两个线程并行运行。

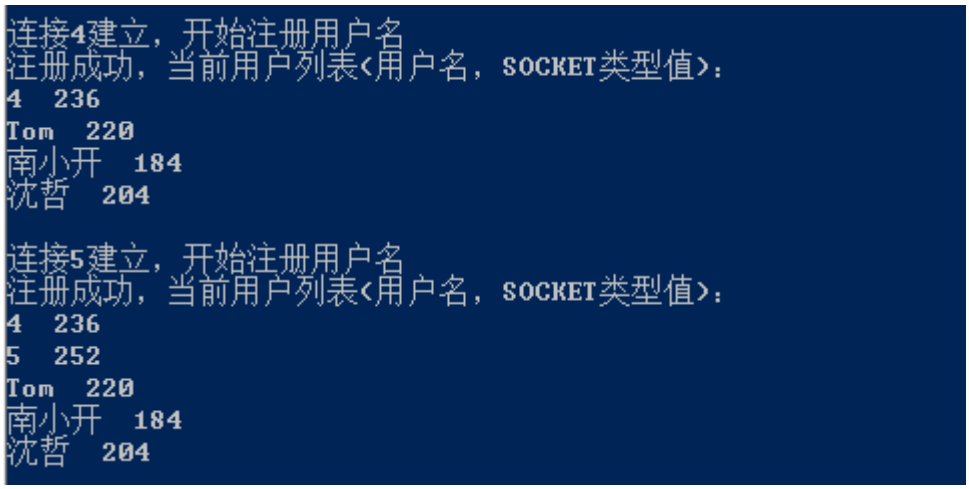
```
CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)receive, (LPVOID)s_server, NULL, NULL);
while (1)
    {
        cout << ">>>";
        char send_buf[100]; cin.getline(send_buf, 100);
        if (!strcmp(send_buf,"exit"))
        {
            return 0;
        }
        int send_len = send(s_server, send_buf, 100, 0);
        if (send_len < 0)
        {
            cout << "发送失败！" << endl;
        }
    }
```

最后关闭套接字释放资源。

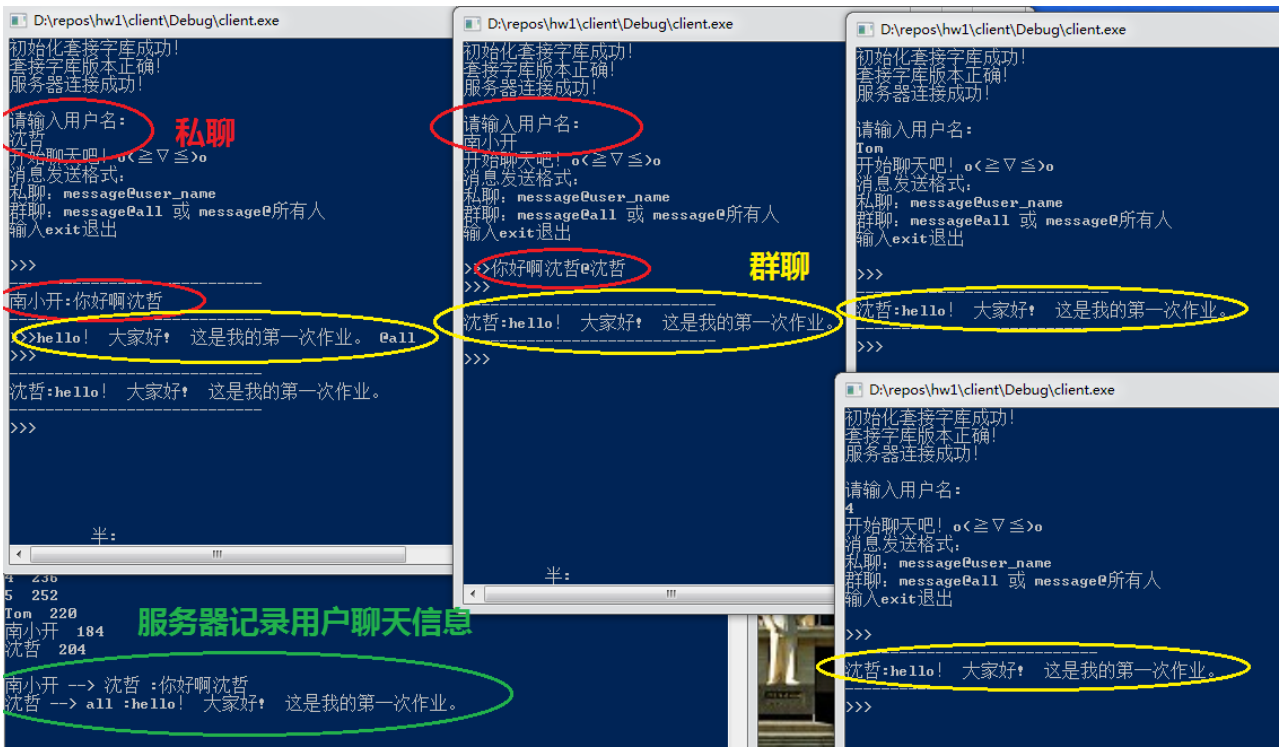
四、结果展示

1.服务器端显示登录用户信息列表：

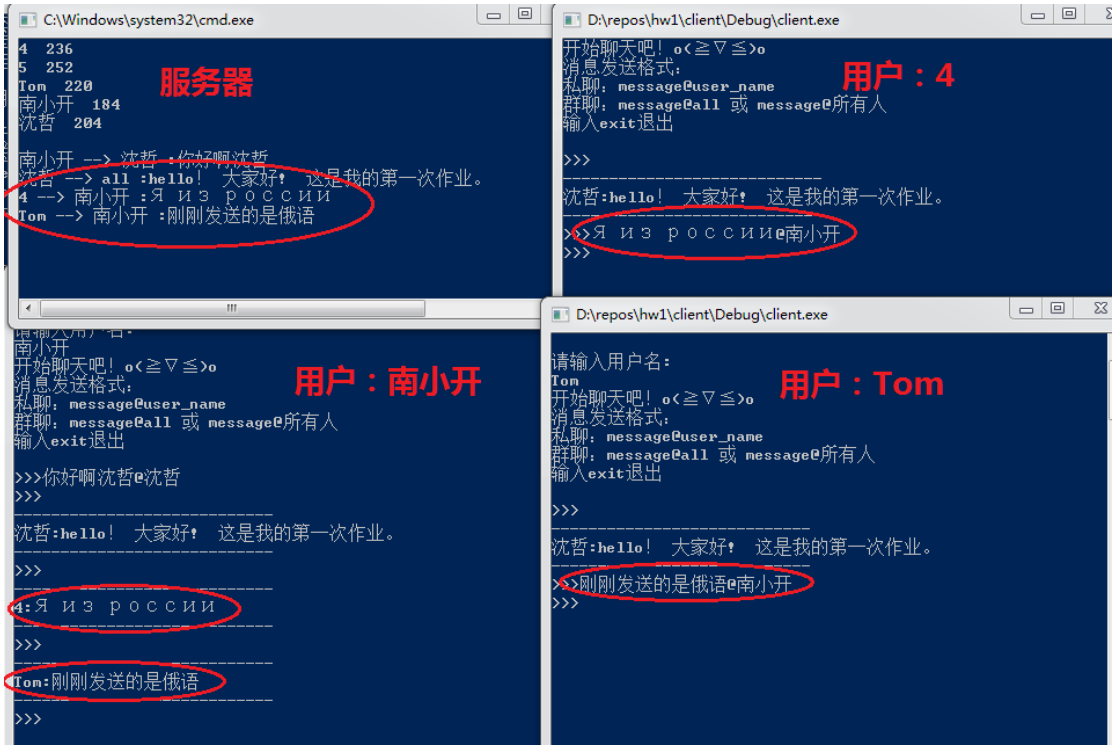




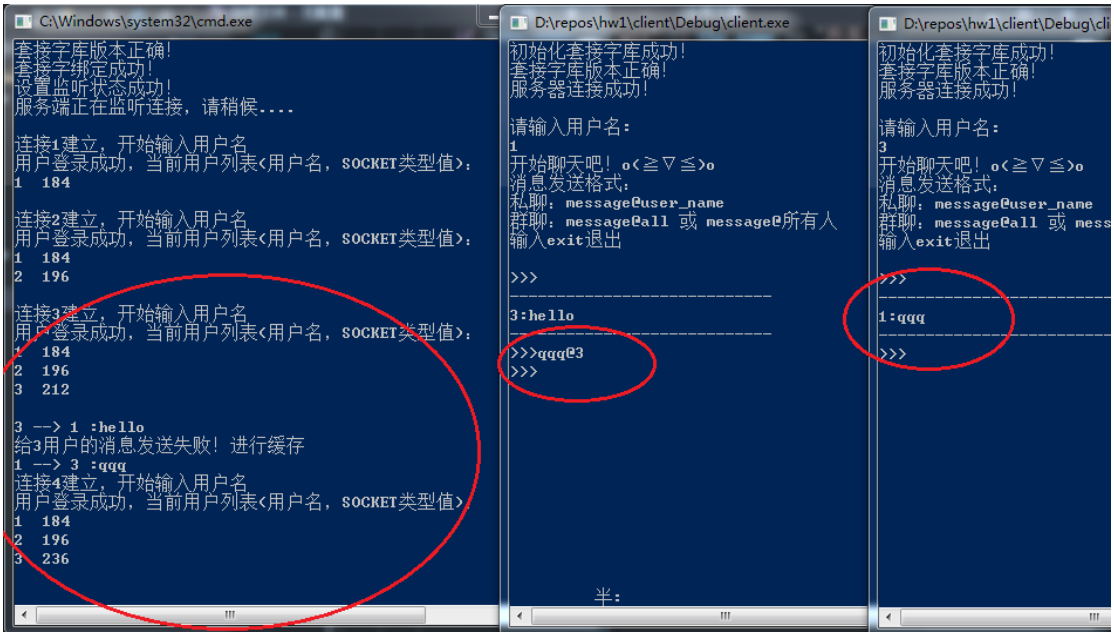
2.私聊和群聊，服务器记录用户聊天信息：



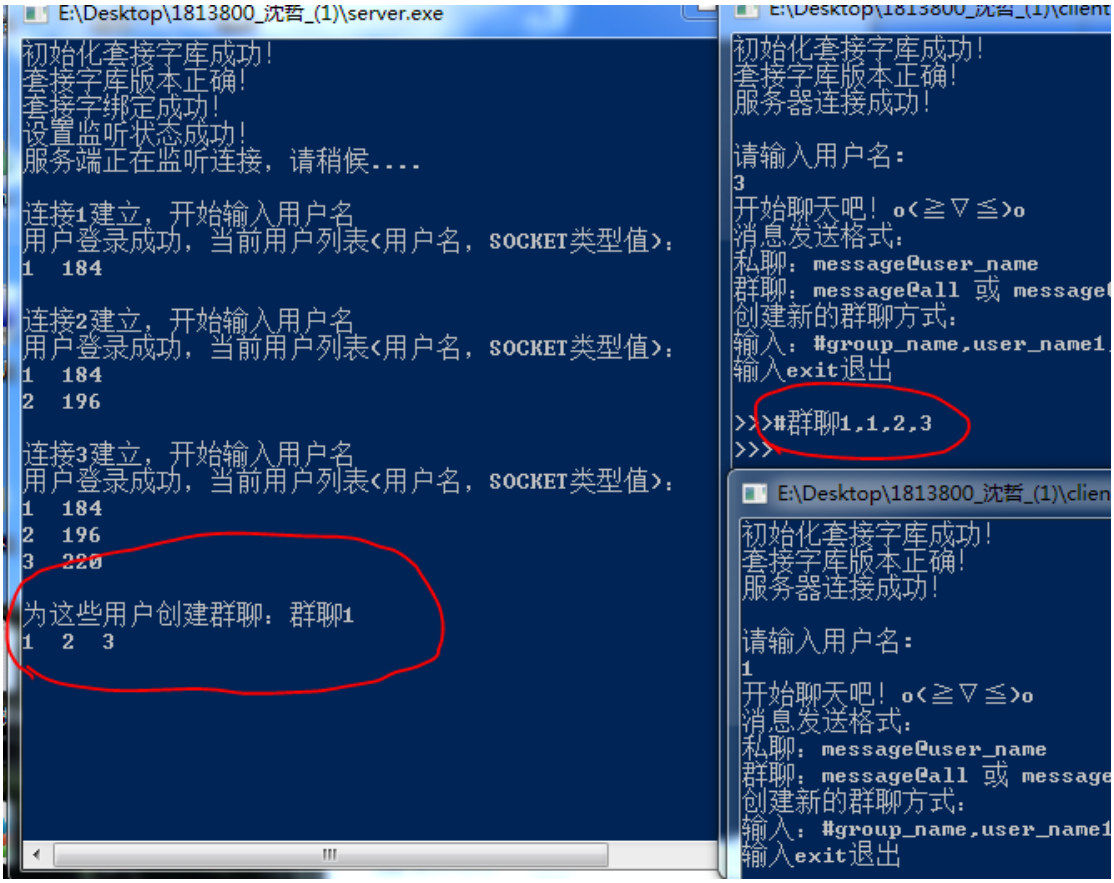
3.支持不同语言，可以看到不同语言也可以混合使用：



4.支持给未在线用户上线后发送消息，用户 1 在用户 3 不在线时发送了一条消息，可以看到用户 3 一登录就收到了用户 1 发的消息：



5.服务器对创建个性化群聊命令的响应：



五、总结展望

1.编程中的一些问题和收获

前期查找服务端和客户端通信、多线程编程等等的资料，花费过多精力，真正开始编写代码时才更加理解。

调试过程中也遇到一些 C++基础知识问题，如：

std 命名空间里有 std::count，自己定义全局变量 count 就会引发冲突。

混合使用 cin 与 getline 函数会出现问题，需要理解缓冲区概念和输入函数的读取过程，做出相应修改。

字符串处理中 string 和 char 有时要进行转换。

经过本次实验，更熟悉了 TCP 协议和 Socket 的基本知识，经过实践对操作系统中进程与线程的概念理解也更深入。

2.不足与改进

本实验从简单的服务端与客户端通信开始，历经几个版本，最终实现不同用户之间的通信，由于时间原因还有很多未能完善。

计划之后可以完善的方向有：

1.完善注册登录环节。增加一个检查，检查新注册名不能与现有用户名重复。

2.完善群聊机制。当前只能算作群发，之后支持创建更小范围的个性化群聊并进行通信。

3.完善消息发送功能。支持更多样的消息，如图片、文件等；支持更灵活的交互方式，如两个用户直接发送消息内容进行通信，不需要附带接收端信息。

4.完善服务器功能，服务器退出时能保证客户端的稳定。

5 在以上基础上，尝试实现图形界面功能。