《网络程序设计》实践课

综合实验报告

实验名称： 网络程序设计综合实验

实验类型： 综合型实验

指导教师： 贾浩

专业班级：

姓　　名：

学　　号：

实验评分

序号	实验分数构成	分数
1	答辩(50 分) （对所设计系统介绍，对教师提问回答情况）	
2	设计思路和详细设计(20 分) （包括实验指导教师提问回答情况）	
3	实验报告书写规范（20 分）	
4	结论和体会（10 分）	

姓名			
实验分数			

基于 C/S 架构的 Windows 平台网络聊天系统设计与实现

【摘要】本实验旨在设计并实现一个基于 Windows 平台的图形化 C/S 结构网络聊天系统。服务端采用 GUI 界面实现对客户端连接的管理，使用 WSAAsyncSelect 模型处理网络事件，并维护用户账户、群聊信息及在线状态，实现广播、私聊和群聊等通信功能。客户端与服务器交互，服务器端通过图形化，与客户端支持在线用户列表更新、消息发送与接收等功能。服务器端在客户端上线或下线时可实时广播通知，同时通过广播按钮手动发布系统消息。系统采用单播方式模拟群聊功能，虽然未实现多播，但设计考虑了多种 I/O 模型的适用性。实验过程中解决了多线程竞争、用户名重复登录等实际问题，并在现有基础上提出了优化私聊与群聊通信机制的方案，体现了对网络编程知识的深入理解与实践能力的提升。

【**Abstract**】This experiment aims to design and implement a graphical C/S architecture network chat system based on the Windows platform. The server employs a GUI interface to manage client connections, utilizes the WSAAsyncSelect model to handle network events, and maintains user accounts, group chat information, and online status to enable communication features such as broadcasting, private messaging, and group chatting. The client interacts with the server through a graphical interface, supporting functions like updating online user lists, sending and receiving messages.The server broadcasts real-time notifications when clients go online or offline, while also allowing manual system messages via a broadcast button. Although multicast communication was not implemented, the system simulates group chat functionality through unicast forwarding, with design considerations for various I/O models. During the experiment, practical issues such as multi-thread contention and duplicate username logins were resolved. Additionally, optimization strategies for private and group chat communication mechanisms were proposed, demonstrating a deep understanding of network programming and enhanced practical skills.

【关键字】C/S 结构，网络聊天，Windows API，WSAAsyncSelect，广播通信，私聊转发，群聊单播

目录

一、问题描述及实验目的

1.1 问题描述

设计并实现一个基于Windows平台的C/S架构网络聊天系统,要求服务端具备用户管理、群组管理、消息转发（广播、私聊、群聊）等功能，客户端支持用户登录、消息发送与接收、在线用户列表更新等交互功能。

1.2 实验目的

（1）熟悉 Windows 平台下套接字编程的基本流程
（2）掌握服务端与客户端通信模型（select、WSAAsyncSelect 等）
（3）掌握多线程模型在服务端的应用
（4）掌握服务器转发机制，实现客户端之间的通信
（5）设计并实现用户管理、广播、群聊等基础功能
（6）掌握通过 GUI 控制服务端的基本方法

二、设计思路

C/S 架构：服务端负责管理用户连接、消息转发和群组维护；客户端通过图形界面与服务器交互。

安全认证：

（1）采用本地文件存储用户凭证（用户名:密码）
（2）密码以明文存储（实验环境简化处理）
（3）登录时验证用户名和密码匹配

通信模型：服务端采用 WSAAsyncSelect 模型处理网络事件，与 GUI 集成，减少线程竞争。客户端采用多线程模型（主线程处理输入，子线程接收消息）。

功能模块：

（1）用户管理：维护在线用户列表，处理登录/注销；
（2）消息转发：支持广播、私聊和群聊（单播模拟）；
（3）群组管理：支持群组创建、加入和消息分发。

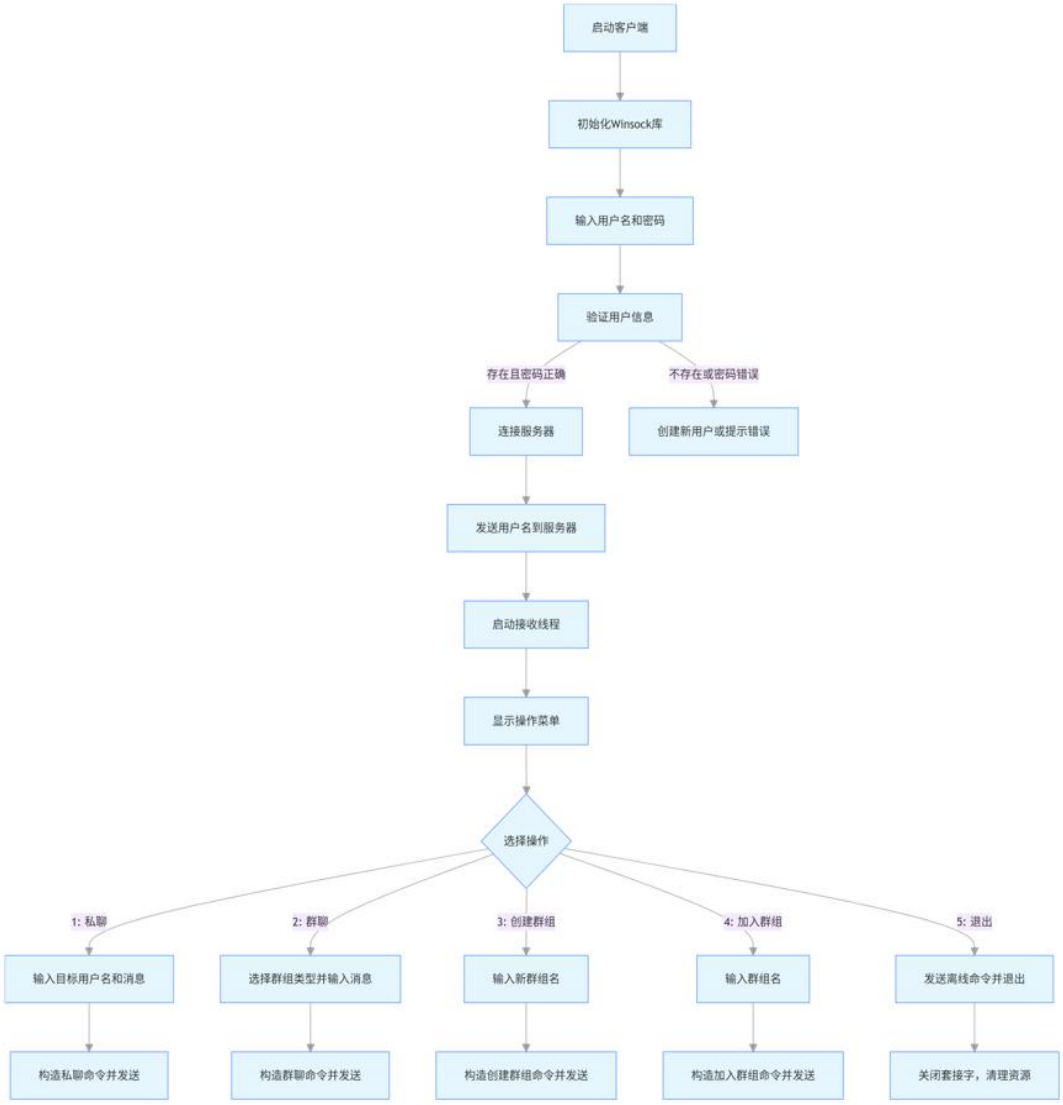


图 1 总体设计流程

三、详细设计过程

3.1 服务端

本系统服务端采用 **图形用户界面（GUI）** 实现对服务器的可视化控制，底层网络通信依托 **Windows API** 完成，主要职责包括用户管理、群组管理、客户端通信转发、广播消息推送等核心功能。

3.1.1 核心功能模块

用户管理

服务端通过维护一个用户结构数组 users[],记录所有登录用户的基本信息及其对应的套接字。每个用户数据项包含用户名、在线状态（active 字段）、连接的 socket 等信息。

数据结构：使用结构体数组 users[MAX_USERS] 存储用户信息（Socket、用户名、在线状态）。

功能实现：

① 当客户端连接成功并登录后，其用户信息会被加入 users[] 中，并设为活跃状态；

② 当用户断开连接或主动退出，服务器立即更新其状态，并通过广播机制通知其他所有在线客户端；

③ 用户上线或下线的消息由服务器生成，作为系统广播统一发布。

④ 用户下线时，清理资源并更新群组成员列表

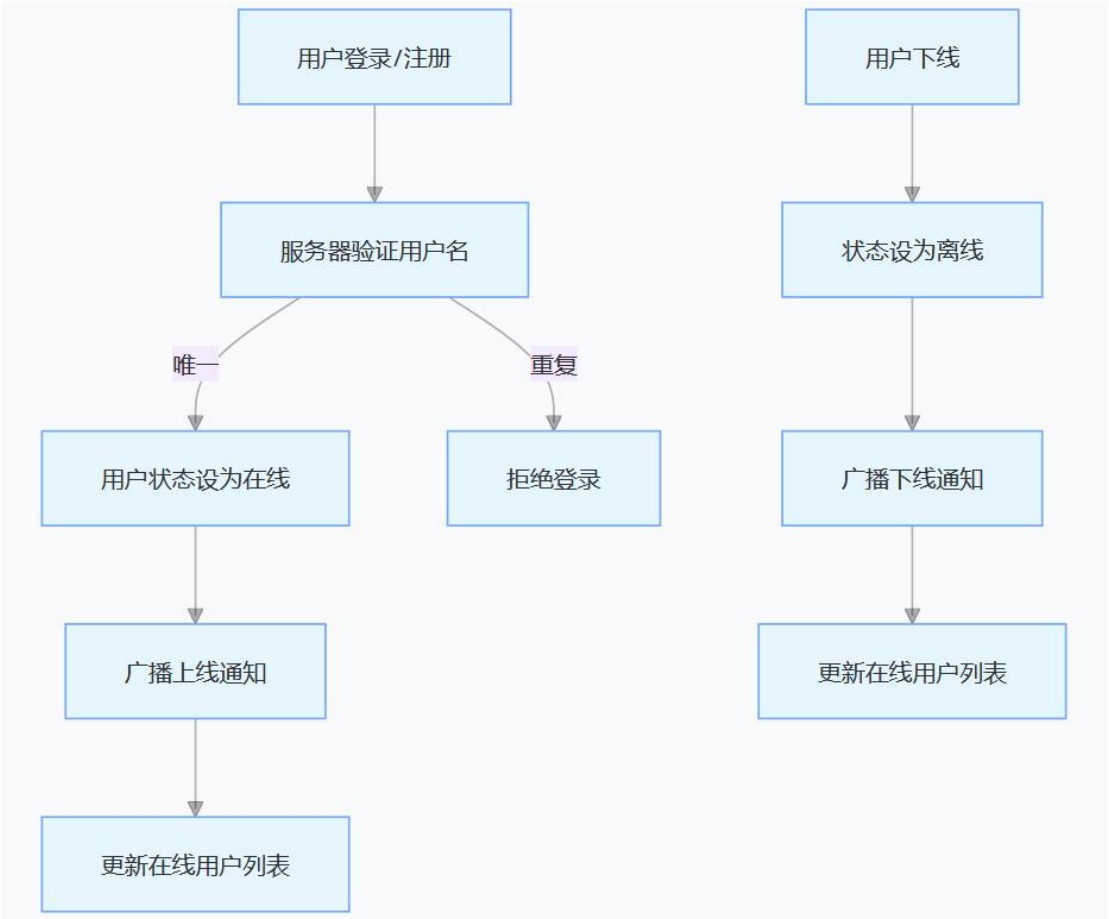


图 2 用户管理流程图

群组管理

服务器支持用户之间的群聊功能，群组的组织采用结构体记录群名、群成员列表等信息。用户可主动加入群组或被服务器添加进群。

数据结构：结构体 Group 存储群组名及成员指针数组。

功能实现：

① 当前版本采用**单播技术**实现群聊，即服务器在处理群聊消息时，将消息逐一发送给群组中的每一位成员；

② 原计划支持多播通信，但多播在实际实现中由于多播组配置和客户端接收机制不完善而未成功实现；

③ [creategroup:群名] 创建群组，发起者自动加入；

④ [joingroup:群名] 将用户加入群组。

图 3 群组管理流程图

消息转发

协议设计：

私聊：[to:用户名]消息内容

群聊：[grp:群名]消息内容

实现功能：

① 当客户端需要向另一个客户发送消息时，它首先将消息发送到服务器，由服务器根据目标用户帐户转发到目标主机。

② 客户端将用户名和消息内容发送到服务器，服务器通过用户名在 user 中找到对应套接字并转发给他（对方要在线）

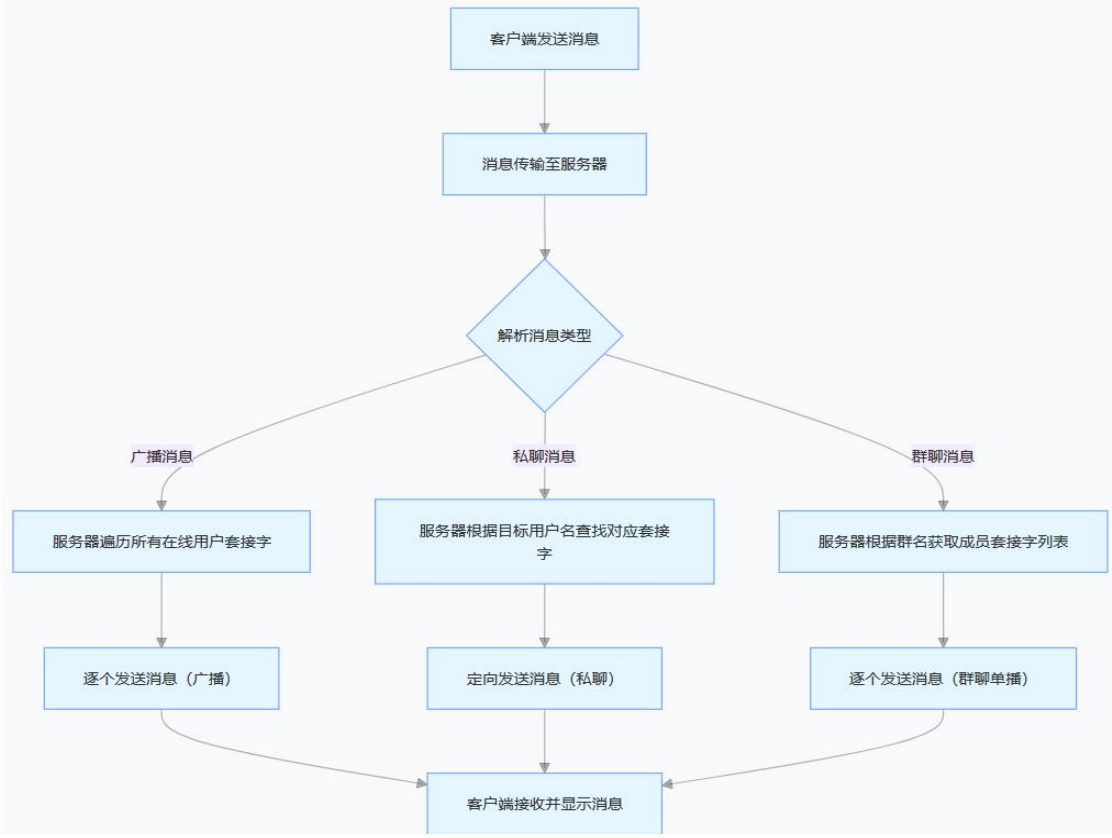


图 4 消息转发流程图

3.1.2 I/O 事件选择模型

WSAAsyncSelect（异步选择） 模型：

服务器带有 GUI 界面，所以 WSAAsyncSelect 可能更适合，因为它能利用窗口消息处理

网络事件，减少线程数。但考虑到高并发需求，可能事件选择模型或完成端口更好。

用户代码已有多线程处理，若改为事件选择模型，可以单线程或少量线程处理多个客户端，减少资源消耗。但需要重新架构主循环，可能改动较大。

先选择 WSAAsyncSelect，因为现有 GUI，可以结合窗口消息处理网络事件，避免多线程竞争,减少线程创建开销（将 Socket 事件绑定到窗口消息循环，避免多线程竞争）。

```
WSAAsyncSelect(serverSocket, hwnd, WM_SOCKET, FD_ACCEPT | FD_CLOSE);
```

3.1.3 客户端通信的套接字实现

客户端之间的通信是通过服务器进行转发的，对于两个客户端，服务器需要创建两个独立的套接字分别维持与客户端之间的连接。

服务端监听套接字：主套接字 serverSocket 绑定端口 8084，监听连接请求。

为每个客户端创建独立套接字：当客户端连接时，accept() 返回一个新的套接字 clientSocket，用于与该客户端通信。套接字存储在 users[] 数组中，关联用户名和状态。

```
users[i].socket = clientSocket;
```

转发消息时的套接字使用：

私聊：通过目标用户名查找对应的 users[i].socket，调用 send() 定向转发。

群聊：遍历群组成员列表，获取每个成员的套接字并发送消息。

3.1.4 GUI 界面

控件：在线用户列表（LISTBOX）、群组列表、广播消息输入框。

功能：手动发送广播消息，实时更新用户/群组列表。

3.2 客户端

3.2.1 核心功能模块

用户登录

连接服务器后发送用户名，等待服务端验证。

```
send(clientSocket, username, strlen(username), 0);
```

消息收发

多线程设计：主线程处理用户输入（菜单选择、消息发送），子线程 RecvThread 持续接收服务器消息并打印。

消息协议

用户输入 1 发送私聊，2 发送群聊，5 退出。

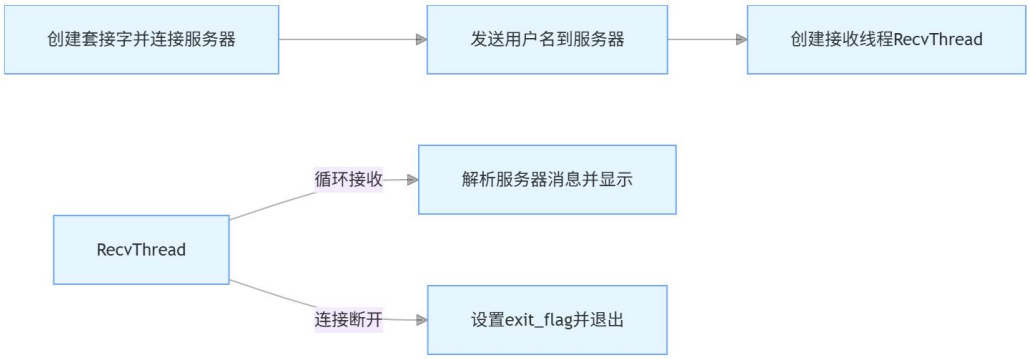


图 5 连接服务器与接收线程的具体流程

密码管理

1. 采用本地文件存储方式，在 users_data.txt 中保存用户名和密码的对应关系
2. 密码以明文存储（实际应用中应加密），格式为用户名:密码
3. 文件读写采用追加模式，新用户注册时自动添加到文件末尾

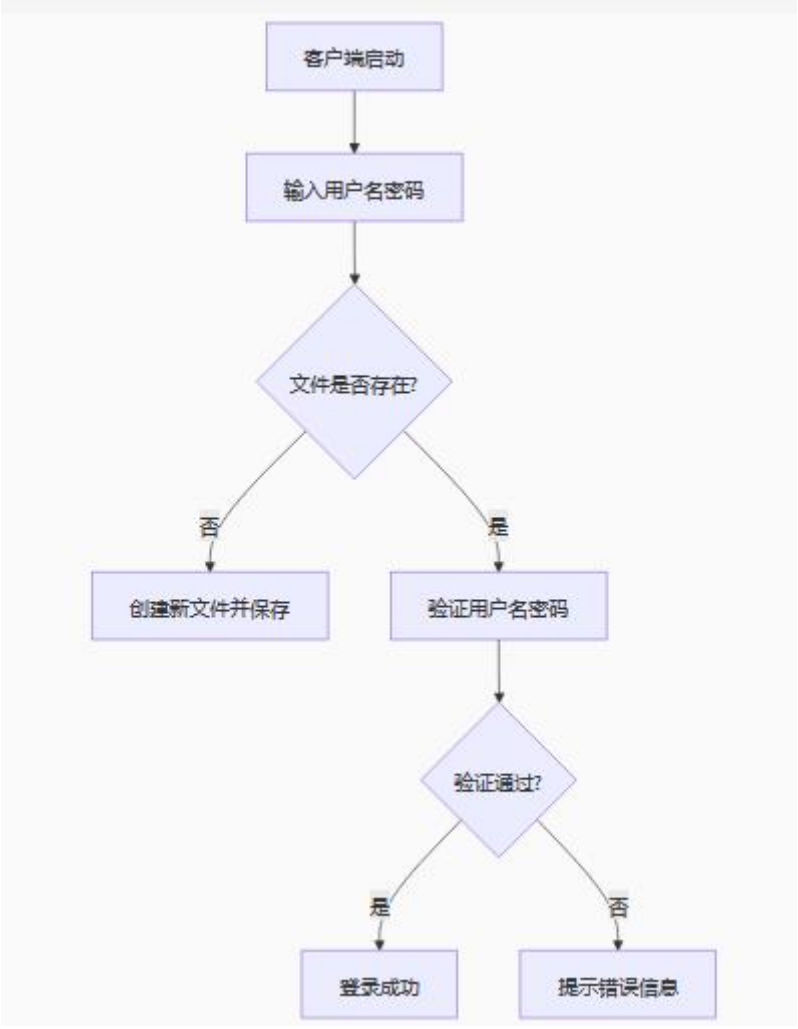


图 6 用户认证与密码管理流程

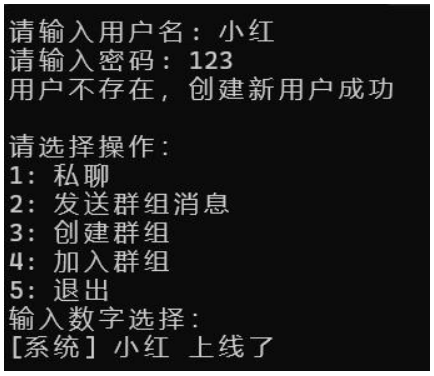
3.2.2 异常处理

断线检测：recv 返回 ≤0 时触发退出。

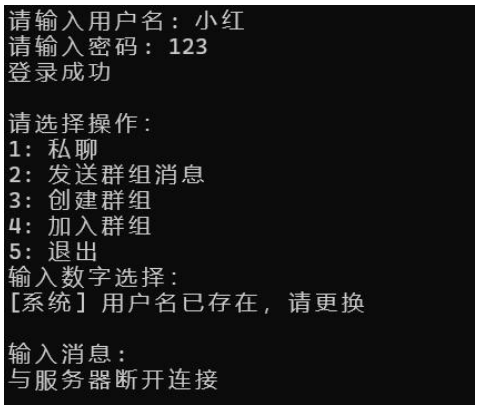
输入校验：防止空消息或无效命令。

四、实验结果

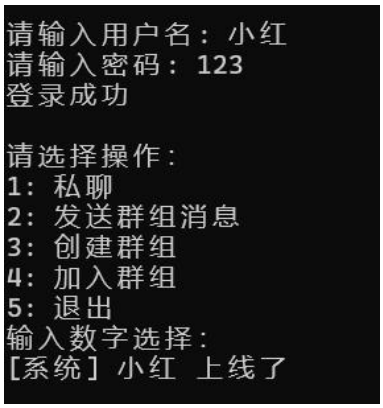
先创建一个用户小红



如果再重复登录，显示用户名已存在，请更换，并与服务器断开连接



小红退出登录后，通过采用本地文件存储方式，在 users_data.txt 中保存用户名和密码的对应关系，再次登录能够登录成功。



再创建一个用户小明

```
请输入用户名：小明
请输入密码：111
用户不存在，创建新用户成功

请选择操作：
1：私聊
2：发送群组消息
3：创建群组
4：加入群组
5：退出
输入数字选择：
[系统] 小明 上线了
```

小红收到小明上线的提醒

```
[系统] 小明 上线了

输入消息：
```

小红私聊小明

```
输入消息：1
输入要私聊的用户名：小明
输入消息内容：hello，小明
```

小明收到私信

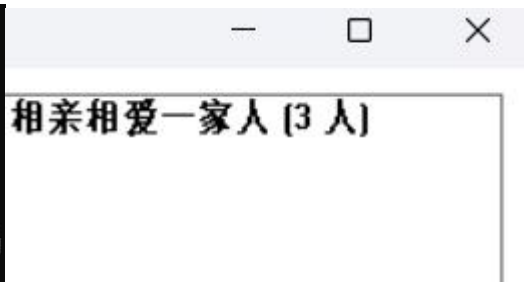
```
[私聊] 小红 to 小明：hello，小明
```

再创建一个用户小美

```
请输入用户名：小美
请输入密码：222
用户不存在，创建新用户成功

请选择操作：
1：私聊
2：发送群组消息
3：创建群组
4：加入群组
5：退出
输入数字选择：
[系统] 小美 上线了

输入消息：
```

小红创建一个群聊

```
输入消息：3
输入新群组名：相亲相爱一家人

请选择操作：
1：私聊
2：发送群组消息

[系统] 群组 '相亲相爱一家人' 创建成功
```



小美小明加入群聊

```
输入消息：4
输入要加入的群组名：相亲相爱一家人

请选择操作：
1：私聊
2：发送群组消息
3：创建群组
4：加入群组
5：退出
输入数字选择：
[系统] 已加入群组 '相亲相爱一家人'
```

都在群里发送消息

```
输入消息：2
请选择要发送消息的群组：
1：全体（广播）
2：输入群组名
输入选择：2
输入群组名：相亲相爱一家人
输入消息内容：Hello!
```

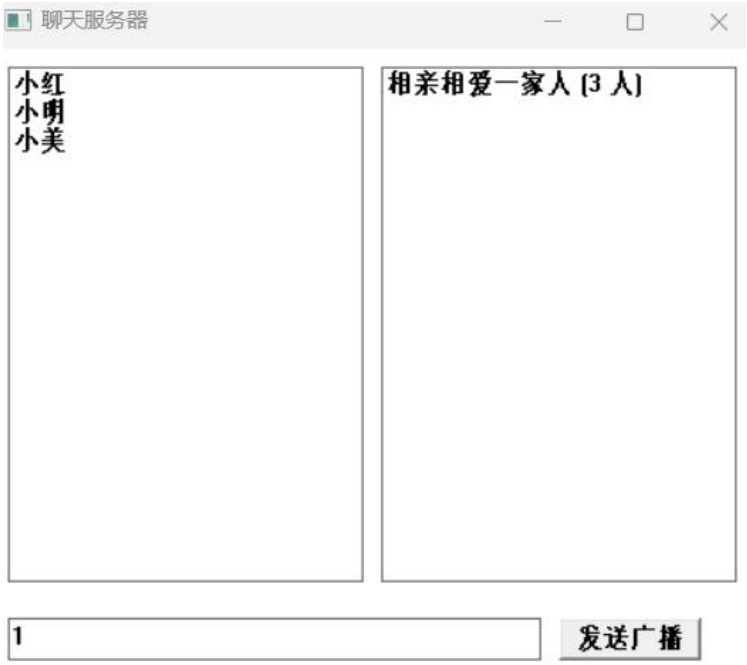
都收到群聊消息

```
输入消息：
[群聊 相亲相爱一家人] 小明：Hello!

输入消息：
[群聊 相亲相爱一家人] 小红：hi
```

小明下线，小红小美收到下线的提醒

```
[系统] 小明 下线了
```

测试发送广播：在聊天框随便输入一条消息选择发送



每个人都接收到广播消息

五、结论及体会

5.1 结论

5.1.1.客户端通信：通过服务器转发

客户端之间的通信完全通过服务端转发完成，具体包括：

① **广播功能**：服务端将任意客户端发送的消息（如系统通知、用户发言）实时转发给所有在线用户。

② **私聊功能**：通过解析消息协议（以[to:]开头），服务端根据目标用户名检索对应套接字，实现一对一消息定向转发（测试中已验证）。

③ **群聊功能**：基于单播技术模拟实现，服务端遍历群组成员列表，逐个发送消息至成员套接字，完成群组内消息分发。

5.1.2 群聊技术实现：单播转发为主

① **当前实现**：群聊依赖单播转发，即服务端将消息按群组成员名单逐个发送（非多播），消息格式为[群聊 群名] 用户名: 内容。

② **未完成目标**：多播通信尝试失败，主要因多播组配置（如客户端加入组播组、防火墙限制）和接收逻辑不完善，导致无法通过多播组一次性广播消息。

③ **技术局限**：单播转发在群组成员较多时会增加服务端负载，效率低于多播。

5.1.3 广播消息：自动广播和手动广播结合

① **自动广播**：用户上线 / 下线时，服务端自动生成系统消息（如[系统] 用户上线），通过广播机制通知所有客户端，并更新在线用户列表。

② **手动广播**：服务端 GUI 界面支持管理员手动输入消息，点击 “广播” 按钮后向全体在线用户发送自定义系统通知。

5.1.4 在线用户信息：实现状态同步和列表更新

① **状态同步**：用户连接或断开时，服务端通过广播消息实时通知所有客户端，确保各客户端的在线用户列表与服务端一致。

② **数据结构**：服务端通过 User 结构体数组维护用户状态（包括用户名、套接字、在线状态），并通过 UpdateUserList 函数更新 GUI 界面的在线用户列表控件。

5.1.5 存在问题与解决方案

问题：用户名重复登录限制异常，即用户下线后，再次使用相同用户名登录时，服务端误判为 “已在线”。

原因：服务器未正确释放掉下线用户的状态，导致再次登录同名用户时判断为“已在线”。即用户断开连接时，未及时将 users[]数组中对应用户的 active 状态标记为 0，导致用户名唯一性验证失败。

解决：在用户下线逻辑中（如套接字关闭事件），遍历 users[]数组，清除对应用户的状态（active=0、释放套接字资源），并调用 UpdateUserList 更新在线列表。

5.2 体会

通过本次实验，深刻理解了 C/S 架构网络通信的核心机制，尤其是在 Windows 平台下如何利用 Socket 编程实现实时交互。

（1） 服务端部分开发体会

服务端采用了图形用户界面（GUI）与基于 Windows Socket API 的网络通信相结合的方式，整合了界面交互与底层通信，设计与开发过程中收获颇多。

首先，在**用户管理模块**的实现中，通过设计结构体数组 users[MAX_USERS] 来维护所有登录用户的状态信息，包括用户名、套接字、在线状态等，切实掌握了用户状态切换、连接/断连处理及广播通知机制的实现。尤其是在处理用户断线或异常退出时，需要及时释放资源并更新用户列表，同时广播下线消息，以维持服务端状态的正确性。这一过程提升了对资源管理、状态同步的掌握。

其次，**群组管理模块**虽然逻辑相对复杂，但提供了良好的抽象和扩展基础。通过设计 Group 结构体管理群组名和成员列表，并实现了 [creategroup:群名] 与 [joingroup:群名] 两种指令格式，初步实现了服务器端对群聊关系的集中控制。使用单播循环转发方式实现群聊功能，让我们更清楚地认识到多播通信在实际部署中的难点与局限。

第三，**消息转发机制**是服务端的核心之一。通过设计协议格式（如 [to:用户名]消息内容、[grp:群名]消息内容）并结合用户/群组信息查找目标套接字，实现了消息定向转发。服务端需要快速准确判断消息类型（私聊/群聊/广播），并基于查找到的目标套接字转发信息。

最后，服务端还承担了**广播消息推送**的任务，例如系统通知用户上线/下线等操作，增强了系统交互性与用户感知。同时，结合 GUI 界面展示用户列表、消息控制台等元素。

（2） 客户端部分开发体会

客户端作为用户直接交互的界面，其设计需要兼顾交互性、稳定性与功能性。本实验中客户端的实现不仅涵盖了登录认证、消息收发、密码管理等基础功能，还引入了多线程机制与异常处理，极大提升了使用体验。

首先，在**用户登录模块**中，客户端在建立与服务器的连接后，通过向服务器发送用户名

请求登录。服务端验证通过后方可进行后续交互。整体结构已经为后续扩展用户认证机制（如口令校验、图形验证码）预留了接口。

其次，**消息收发机制**是客户端的核心功能。在设计中，采用了主线程 + 接收线程（RecvThread）的多线程模型：主线程处理用户输入与命令解析，接收线程持续监听服务器推送消息并异步显示。这种非阻塞式设计有效解决了用户输入与消息接收之间的冲突问题，提升了程序的响应速度与用户体验

第三，**消息协议设计**采用数字编号与关键字控制的方式，使得用户输入更加简洁明确，例如输入“1”进入私聊，“2”进入群聊，“5”表示退出等。这种协议虽然简单，但逻辑清晰，为后续功能拓展（如好友请求、文件传输等）提供了良好的可扩展性。

在**密码管理模块**中，客户端通过本地 users_data.txt 文件记录用户名与密码的对应关系，采用文本方式存储，便于快速读写和调试。新用户注册时自动追加信息至文件末尾。当前版本为简化开发采用明文存储密码。

在**异常处理方面**，客户端具备基本的容错机制。例如，在连接过程中若检测到 recv() 返回值小于等于 0，则自动判定服务器断开连接并退出程序；同时在用户输入环节加入了输入校验逻辑，有效避免了空消息或非法指令引发的异常。通过这些细节的处理，提高了系统的健壮性。

综上，客户端开发让我们系统掌握了基于 Socket 的客户端交互流程、多线程并发模型、简单协议设计与文件操作等关键技术，同时也意识到用户体验优化与安全保障的重要性，为未来构建更加成熟的客户端系统打下了坚实基础。

通过本次综合实验，不仅实现了一个功能完善的网络聊天系统，更在实践中加深了对 C/S 通信模型、套接字编程、Windows API 及多线程模型等核心知识点的理解。实验过程中，针对多客户端连接、消息转发与同步、图形界面交互等复杂问题进行了深入思考与优化，有效提升了分析问题和解决问题的能力。在系统实现与调试的过程中，锻炼了代码结构设计能力和工程实现能力，为今后从事网络编程相关工作或研究打下了坚实的基础。整体而言，本次实验不仅验证了课堂所学理论知识，更增强了动手能力和团队协作意识，达到了综合性实验的预期目标。

六、附录

附录 A 服务端关键代码

核心数据结构

```
// 用户结构体
typedef struct {
    SOCKET socket;
    char username[50];
    int active;              // 在线状态标志
    int authenticated;    // 认证状态标志
} User;//User 结构体存储用户信息，包括套接字、用户名和在线状态
// 群组结构体
typedef struct {
    char groupName[50];
    User* members[MAX_USERS]; // 群组成员指针数组
    int count;               // 当前成员数
```

```
} Group;//Group 结构体记录群组名和成员列表。

// 全局变量
User users[MAX_USERS] = { 0 };  // 用户数组
Group groups[MAX_GROUPS] = { 0 }; // 群组数组
int userCount = 0, groupCount = 0; // 计数变量
```

用户管理功能

```
// 检查用户名是否存在
int UsernameExists(const char* username) {//UsernameExists 用于防止用户名重复登录。
    for (int i = 0; i < MAX_USERS; i++) {
        if (users[i].active && strcmp(users[i].username, username) == 0) {
            return 1;
        }
    }
    return 0;
}
```

```
// 更新用户列表控件
void UpdateUserList(HWND hwnd) {//UpdateUserList 通过窗口消息更新 GUI 中的在线用户列表控件
    SendMessage(hUserList, LB_RESETCONTENT, 0, 0);
    for (int i = 0; i < MAX_USERS; i++) {
        if (users[i].active && users[i].authenticated) {
            SendMessage(hUserList, LB_ADDSTRING, 0, (LPARAM)users[i].username);
        }
    }
}
```

群组管理功能

```
// 查找群组
Group* FindGroup(const char* groupName) {//FindGroup 通过群组名快速定位群组
    for (int i = 0; i < MAX_GROUPS; i++) {
        if (groups[i].count > 0 && strcmp(groups[i].groupName, groupName) == 0) {
            return &groups[i];
        }
    }
    return NULL;}
// 更新群组列表（GUI）
void UpdateGroupList(HWND hwnd) {//UpdateGroupList 动态更新 GUI 中的群组列表。
    SendMessage(hGroupList, LB_RESETCONTENT, 0, 0);
    for (int i = 0; i < MAX_GROUPS; i++) {
        if (groups[i].count > 0) {
            char groupInfo[100];
            sprintf_s(groupInfo, "%s (%d 人)", groups[i].groupName,
groups[i].count);
            SendMessage(hGroupList, LB_ADDSTRING, 0, (LPARAM)groupInfo);
```

```
        }
    }
}
```

消息转发功能

```
// 广播消息给所有在线用户
void SendBroadcast(const char* message) {//SendBroadcast 实现系统广播功能。
    for (int i = 0; i < MAX_USERS; i++) {
        if (users[i].active && users[i].authenticated) {
            send(users[i].socket, message, strlen(message), 0);
        }
    }
}

// 处理私聊消息
if (strstr(buffer, "[to:") != NULL) {
    char target[50], content[BUFFER_SIZE];
    sscanf_s(buffer, "[to:%49[^]]]%1023[^\n]", target, 50, content, BUFFER_SIZE);
    for (int i = 0; i < MAX_USERS; i++) {
        if (users[i].active && strcmp(users[i].username, target) == 0) {
            char msg[BUFFER_SIZE];
            sprintf_s(msg, "[私聊] %s: %s\n", currentUser->username, content);
            send(users[i].socket, msg, strlen(msg), 0);
        }
    }
}
```

网络事件处理（异步选择模型）

```
// 绑定 Socket 事件到窗口消息
WSAAsyncSelect(serverSocket, hwnd, WM_SOCKET, FD_ACCEPT | FD_CLOSE);
// 处理 Socket 事件
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    switch (msg) {
    case WM_SOCKET:
        HandleSocketEvent(hwnd, wParam, lParam);
        break;
    case WM_CREATE: {
        hUserList = CreateWindow("LISTBOX", "",
            WS_VISIBLE | WS_CHILD | LBS_NOSEL | WS_BORDER | WS_VSCROLL,
            10, 10, 200, 300, hwnd, (HMENU)IDC_USER_LIST, NULL, NULL);

        hGroupList = CreateWindow("LISTBOX", "",
            WS_VISIBLE | WS_CHILD | LBS_NOSEL | WS_BORDER | WS_VSCROLL,
            220, 10, 200, 300, hwnd, (HMENU)IDC_GROUP_LIST, NULL, NULL);

        hBroadcastEdit = CreateWindow("EDIT", "",
```

```
            WS_VISIBLE | WS_CHILD | WS_BORDER | ES_AUTOHSCROLL,
            10, 320, 300, 24, hwnd, (HMENU)IDC_BROADCAST_EDIT, NULL, NULL);

        HWND hBroadcastBtn = CreateWindow("BUTTON", "发送广播", WS_VISIBLE |
WS_CHILD,
            320, 320, 80, 24, hwnd, (HMENU)IDC_BROADCAST_BUTTON, NULL, NULL);

        UpdateUserList(hwnd);
        UpdateGroupList(hwnd);
        break;
    }
    case WM_COMMAND:
        if (LOWORD(wParam) == IDC_BROADCAST_BUTTON) {
            char msg[BUFFER_SIZE];
            GetWindowTextA(hBroadcastEdit, msg, BUFFER_SIZE);
            if (strlen(msg) > 0) {
                char broadcastMsg[BUFFER_SIZE];
                sprintf_s(broadcastMsg, "[服务器广播] %s\n", msg);
                SendBroadcast(broadcastMsg);
                SetWindowTextA(hBroadcastEdit, "");
            }
            break;
        }
        break;
    case WM_USER_UPDATE_GROUP_LIST:
        UpdateGroupList(hwnd);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

附录 B 客户端关键代码

用户登录与认证

```
// 发送用户名到服务端
send(clientSocket, username, strlen(username), 0);

// 接收线程处理服务端响应
unsigned __stdcall RecvThread(void* lpParam) {
```

```
    char buffer[BUFFER_SIZE];
    while (!exit_flag) {
        int recvSize = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (recvSize <= 0) {
            printf("\n 连接已断开\n");
            exit_flag = 1;
            break;
        }
        buffer[recvSize] = '\0';
        printf("\n%s", buffer);
    }
    return 0;
}
```

消息发送逻辑

```
// 私聊消息格式
sprintf_s(command, "[to:%s]%s", targetUser, message);
send(clientSocket, command, strlen(command), 0);

// 群聊消息格式
sprintf_s(command, "[grp:%s]%s", groupName, message);
send(clientSocket, command, strlen(command), 0);
```

密码管理

```
FILE* file;
errno_t err = fopen_s(&file, "users_data.txt", "a+");
if (err != 0) {
    printf("无法打开用户数据文件，创建新用户\n");

    // 创建新用户
    err = fopen_s(&file, "users_data.txt", "w+");
    if (err != 0) {
        printf("无法创建用户数据文件，错误代码：%d\n", errno);
        WSACleanup();
        return 1;
    }

    fprintf(file, "%s:%s\n", username, password);
    fclose(file);
    WSACleanup();
    return 0;
}
else {
    // 查找用户名并验证密码
    char line[100];
    int found = 0;
```

```
    // 确保文件指针移动到文件开头
    fseek(file, 0, SEEK_SET);

    while (fgets(line, sizeof(line), file)) {
        // 跳过空行
        if (line[0] == '\n' || line[0] == '\r') {
            continue;
        }
        #define MAX_USERNAME_LEN 50

        char fileUsername[MAX_USERNAME_LEN] = { 0 };
        char filePassword[50] = { 0 };

        // 检查 line 是否足够解析
        if (strlen(line) < 3 || strchr(line, ':') == NULL) {
            printf("跳过格式错误的行：%s\n", line);
            continue; // 跳过格式明显错误的行
        }

        // 使用 sscanf_s 并指定缓冲区大小
        if (sscanf_s(line, "%49[^:]:%49s", fileUsername,
(unsigned)MAX_USERNAME_LEN, filePassword, (unsigned)50) != 2) {
            printf("解析失败，当前行内容：%s\n", line);
            continue; // 如果解析失败，跳过这一行
        }

        if (strcmp(fileUsername, username) == 0) {
            found = 1;
            if (strcmp(filePassword, password) == 0) {
                printf("登录成功\n");
            }
            else {
                printf("密码错误\n");
            }
            break;
        }
    }

    if (!found) {
        printf("用户不存在，创建新用户成功\n");

        // 将文件指针移动到文件末尾
        fseek(file, 0, SEEK_END);
```

```
            // 尝试将新用户写入文件
            if (fprintf(file, "%s:%s\n", username, password) <= 0) {
                printf("写入新用户数据失败\n");
                fclose(file);
                WSACleanup();
                return 1;
            }
        }

        fclose(file); // 在所有操作完成后关闭文件
    }
```

异常处理

```
// 断线检测
if (recvSize <= 0) {
    printf("\n 连接异常\n");
    closesocket(clientSocket);
    exit_flag = 1;
}//接收失败时触发退出标志，清理 Socket 资源。

// 资源清理
if (exit_flag) {
    closesocket(clientSocket);
    WSACleanup();
}
```

七、参考文献

[1] 杨秋黎 金智，《Windows 网络编程（第 2 版）》，人民邮电出版社，2015.1
[2] 张银奎，《Windows 网络编程与 WinSock 实践》，清华大学出版社,2012.1
[3] 李洪发，《Windows 网络编程实例解析》，电子工业出版社,2016.11