北　京　邮　电　大　学

项目概要设计

课程名称　　程序设计综合实验

项目名称　　语你--实时通讯系统

人工智能学院 2020219108 班

成员一：杨再俨　学号：2020212183

成员二：王焕捷　学号：2020212181

成员三：张宏伟　学号：2020212429

2022 年 5 月 4 日

姓名：杨再俨 学号：2020212183
姓名：王焕捷 学号：2020212181
姓名：张宏伟 学号：2020212429

1.系统概述

1.1 系统简介

现代社会人们已经习惯与他人进行线上交流，近些年随着疫情的到来，线上办公、实时通信也成为热门应用领域。我们欲开发一个聊天软件，该软件能够实现用户登录、好友管理、实时通信、查找聊天记录等基础功能。该软件可以看作简易版的聊天工具。我们的目标是实现类似QQ、微信聊天软件的基本功能，于此同时又使得该软件专注于聊天。软件整体体积较小，功能简洁明了，使用方便、上手容易。我们希望在开发中加强自己对C/C++语言的运用能力，同时巩固所学的计算机网络、网络编程以及项目设计知识点的掌握。

传统的软件开发生命周期的主要步骤是需求分析、概要设计、实现和测试。如果说需求分析阶段提供了一个对预期软件产品的描述，那么概要设计的主要任务是把需求分析得到的系统扩展用例图转换为软件结构和数据结构。设计软件结构的具体任务是：将一个复杂系统按功能进行模块划分、建立模块的层次结构及调用关系、确定模块间的接口及人机界面等。数据结构设计包括数据特征的描述、确定数据的结构特性、以及数据库的设计。显然，概要设计建立的是目标系统的逻辑模型，与计算机无关。概要设计主要是为预期系统的构建提出一个计划，即制定在需求分析阶段描述的问题的解决方案。在软件的设计中画形化和建模发挥着很大的作用，下面简单介绍一下软件的开发和设计过程中使用的建模技术和符号系统。

i.**结构图**专门用于命令型范型（类似于使用C语言进行程序设计），命令型范型致力于依据过程或者函数来构建软件，故需要研究数据在系统中如何流动是一大重点，常用**数据流图**表示从数据流分析过程中所获得的信息，构建数据流图可以有效改善客户与软件工程师之间的交流。

ii.**数据字典**是另一种常用工具，构建数据字典以保证整个系统的统一性，避免因为每个程序员的编写习惯不同导致出现变量名冲突等问题。

1.2 系统目标

系统须实现的功能主要如下：

1. 操作者能够通过用户名和密码注册用户
2. 已注册用户登录
3. 添加和删除好友
4. 列出所有的好友用户
5. 查询所有的在线用户
6. 查询好友状态
7. 发送消息给好友
8. 帮助菜单
9. 发送消息
10. 查询聊天记录
11. 显示用户信息
12. 找回密码
13. 用户退出

在系统的性能方面，系统暂定能够支持注册的用户数量为 200 人，若注册人数超过该限制会显示注册失败。限制注册人数是为了保证不给服务器造成过大的负载。服务器使用多线程的模式保证多个用户能够同时在线并与好友进行交流沟通。同时系统引入文件模块来保存用户信息、聊天信息等。考虑到随着使用人数的不断增加，后期使用文件模块进行信息的存储将不是一种可行的方法，同时把一些不是特别重要的信息保存在服务器端也会造成不必要的浪费，可以考虑将这些信息存储在客户端，而服务器端使用数据库等专用的数据管理系统进行管理。

1.3 系统运行环境

程序运行环境:只能为 Windows 系统
硬件平台:个人计算机
数据库系统:无要求
网络协议:TCP 面向连接的可靠协议

编程平台:无要求，编程平台只用来生成 exe 文件

我们设计的程序为 windows 操作系统下的 exe 可执行程序，运行本程序需在 windows 命令行窗口下输入相关的命令行指令运行。

1.4 开发环境

开发环境

i. 操作系统：Windows10/Windows11

ii. 编程语言：C/C++

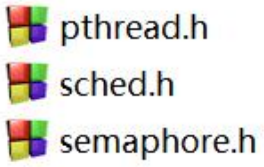
iii. 开发环境：Visual Studio 2022 集成开发环境 / CodeBlock-17.12 / CodeBlock-20.03

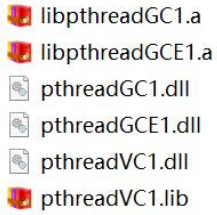
该系统运行以及代码的实现都是在 Windows 操作系统下完成的，尽管涉及到网络编程模块，但是考虑到 Windows 系统对于用户友好的使用体验，故我们选择了 Windows 操作系统。由于在设计初期我们并没有考虑使用数据库来对系统的信息进行管理，所以这里不赘述。

编写代码的工具可以选择纯文本编辑器如 VsCode 或 notepad 等文本编辑器，为了方便调试和运行，建议使用集成 IDE 如 Visual Studio 或 Code Blocks。编程语言我们选择了 C 语言进行编程，一方面是为了巩固我们对 C 语言的掌握能力，一方面是考虑到网络编程环境下 C 语言的灵活性。

因为编程习惯的不同，我们小组选择了不同的集成 IDE 进行代码的编写，在此期间考虑到使用到 Git 对文件进行管理，考虑到组员对于工具的使用，最终还是决定手工进行文件的整理归纳。因为都是在 Windows 操作系统下使用 C 语言进行系统的设计，所以在程序代码的沟通方面并没有出现问题。唯一需要注意的点就是涉及网络编程模块的一些库可能需要手动设置动态链接或者自己下载导入。

为了实现多线程，我们需要手动配置几个文件如下





在客户端与服务器的交互过程中涉及到应用层进程之间的交互，我们使用 TCP 面向连接的可靠协议来提供可靠的传输服务。

2. 总体结构设计

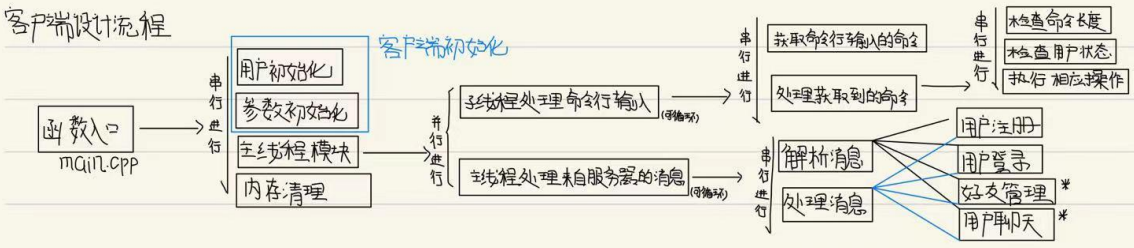
实时通信系统主要分为两大部分——客户端和服务器部分，每个部分分别由不同的小模块组成。这里我们不区分客户端和服务器，先简要列出该系统最重要的几个模块，搭建程序的基本结构。

模块名	主要功能
用户注册模块	用户通过客户端进行注册，服务器返回用户唯一的 id 和注册使用的密码
用户登录模块	用户通过客户端进行登录，服务器进行用户验证并返回登录认证
好友管理模块	用户通过客户端发起对好友进行管理的请求（添加、删除）
消息管理模块	客户端发送消息给服务器；服务器响应消息给客户端
信息存储模块	存储、管理聊天记录、用户信息等
网络编程模块	客户端和服务器之间一切涉及网络传输的行为
指令处理模块	用于解析用户从命令行输入的指令

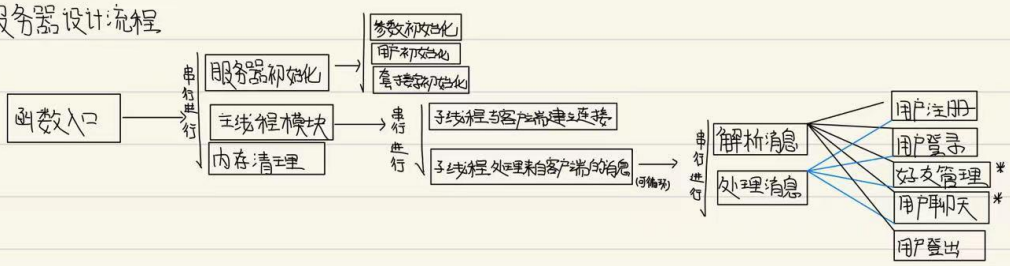
2.1 软件结构

下面以图形的方式给出了软件系统的子系统划分、模块间的关系、软件设计流程图以及各模块之间的松散耦合关系

客户端设计流程



服务器设计流程




用户注册
用户登录认证
好友管理
网络编程
聊天信息管理
消息发送
信息分装和获取

耦合关系	用户注册	用户登录认证	消息发送	信息分装和获取	聊天信息管理	好友管理	网络编程
用户注册							
用户登录认证	内容耦合						
消息发送	非直接耦合	非直接耦合					
信息分装和获取	非直接耦合	控制耦合	控制耦合				
聊天信息管理	非直接耦合	非直接耦合	数据耦合	控制耦合			
好友管理	非直接耦合	非直接耦合	非直接耦合	非直接耦合	非直接耦合		
网络编程	数据耦合	数据耦合	数据耦合	数据耦合	数据耦合	数据耦合	

2.2 设计思想

设计思想：

我们采用“瀑布模型”进行软件设计，其核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作。不难看出，我们将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。该开发思想的优点是：将开发阶段通过一系列过程展开，每个阶段都可产生循环反馈，并能够及时发现问题返回上一阶段进行修改。例如，在程序编写阶段，若在实现之前所设计的模块时遇到困难，可返回上一阶段重新设计，能够很好的保证程序设计和效率，提高设计成功率。

设计理念：

1.我们的程序服务于了解计算机基本操作的人，只需简单的打开程序，注册账号，便可与他人进行实时通讯。在互联网、计算机更加普及的未来，我们的程序可拥有更多的操作用户，具备良好的发展前景。

2.我们的程序应用功能较为强大，其明确目的就是通过其实用性和简洁性吸引用户使用我们的程序进行通信交流

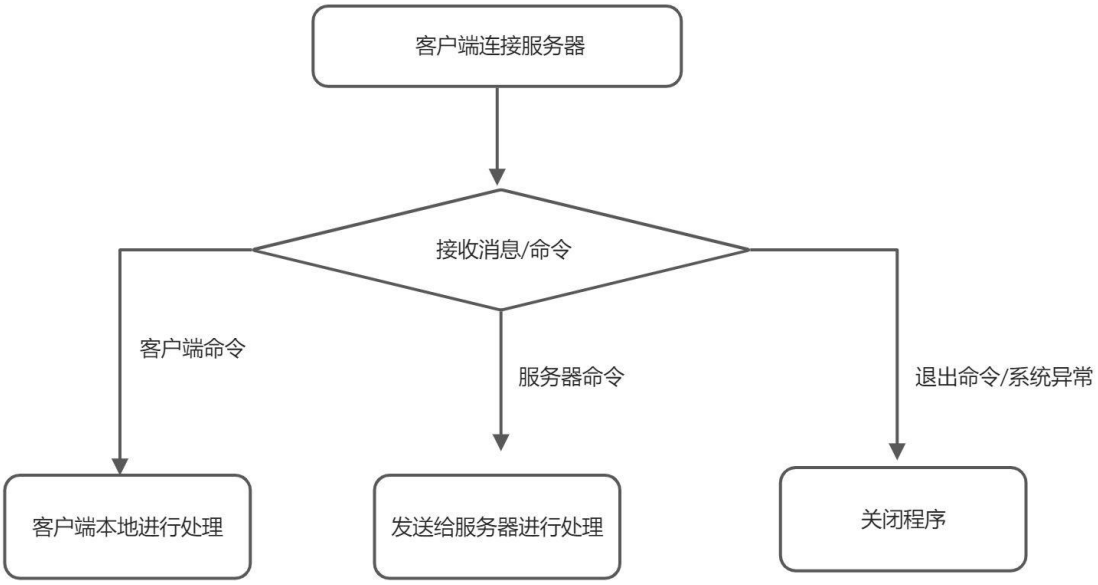
3.我们的程序可解决的问题：在确保安全性的情况下，快速完整的实现用户间的信息交流。
4.关于定义交互方式，我们将基于 QT 设计优美简洁的用户界面使交互变的更加灵活，通过定义点击、滑动等操作，或将不同的功能，不同的界面呈现给用户。
5.关于储存，我们初步设计以管理员主机为内存来构建数据模型。未来将以数据库形式实现数据存储，具有良好的可扩展性、可优化性。

3.模块设计

3.1 模块 1：客户端

文件名	对应模块	功能简介
client_config.h	客户端配置	客户端常用配置文件
client_handle .h/.c	消息处理	客户端消息处理程序
client_log .h/.c	日志管理	客户端日志管理
client_user .h/.c	好友管理	客户端好友管理
cmsg_dec .h/.c	消息解析	对收到的消息进行解析
cmsg_enc .h/.c	消息封装	对要发送消息进行封装
cusr_cmd .h/.c	用户指令处理	处理用户输入的指令
im_client.h	无	包含所有模块的头文件
list .h/.c	链表库	对链表的操作（好友管理）
msg_cipher .h/.c	聊天信息加解密	聊天消息加密和解密
msg_type.h	消息类型	消息类型及结构体定义
client_main.c	客户端主程序	客户端主程序

3.1.1 功能描述



3.1.2 接口描述

首先从 main 函数入手，它是整个客户端的入口。我们从 main 函数中接收用户输入的参数，对整个客户端进行初始化（包括套接字的创建、用户参数、用户结构体的初始化...），初始化完成后我们就可以进入一个主循环来进行和服务器的连接通信等，在该循环中我们创建子线程，子线程中我们持续监听指令，判断下一步客户端需要执行的操作（这里会用到用户处理指令模块）。假如得到退出指令则我们断开客户端和服务器之间的连接并退出客户端程序。系统需要在用户关闭套接字后对套接字进行清理以释放内存。

在主循环中会涉及几乎所有模块。当客户端进行注册时需要涉及注册消息的封装以及消息的发送（cusr_cmd .h/.c，cmsg_enc .h/.c），当客户端处理用户指令时需要使用到用户指令、消息处理模块（client_handle .h/.c）。当客户端给另一个客户端发送消息的时候先给要发送的消息进行加密，
（msg_cipher.h/.c 模块）接着是发送给服务器进行转发，服务器在收到来自客户端的消息后对其进行解析和转发，发送给另一个客户端。当另一个客户端收到该聊天消息后需要进行对消息的提取、对聊天消息的解密(msg_cipher .h/.c)。

3.1.3 数据结构描述

（1）本模块使用了双向循环链表来实现好友的管理，这里的相关的操作与数据结构里面的双向循环链表方法一致，即为有指向上一个节点的指针和指向下一个节点的指针且该链表的尾部指向该链表的头部。

（2）本模块使用了循环遍历来进行消息（包括账号和密码）的加密与解密，加密与解密的方式是遍历消息的每一个字符然后判断是否要加上 key 值，解密以相反的操作减去 key 值。

（3）本通讯系统的客户端使用了很多有关 socket 的函数来实现网络编程，在实现该客户端的时候，客户端需要先建立 socket，然后在连接至服务器端之后进行发送消息和接受消息，最后在使用完毕之后进行关闭。这里使用得到数据结构是有关网络编程的大量函数。

（4）以下是客户端与服务器端的交互的消息的类型，同样也是客户端使用的消息数据结构。

消息类型	消息子类型	简介
注册消息	无	用户注册向服务器发起注册
登录认证消息	无	用户登录认证消息
好友管理消息	好友状态查询	查询该用户是否在线
	好友列表请求	客户端服务器请求好友列表
	增加好友	客户端服务器请求增加好友
	删除好友	客户端服务器请求删除好友
	列出所有用户	客户端服务器请求列出所有用户
聊天消息	无	用户向其他用户发生聊天信息
退出登录	无	用户向服务器发起退出登录消息

3.1.4 实现思路

利用 C 语言原有的库以及从 powershell 窗口打开时候可以输入多个 main() 函数参数来进行对整个客户端的实现，客户端的所有的函数以及头文件都写在同一个目录下面，在打开客户端的时候就在该文件目录下面进行操作即可。对于客户端内部的各个功能（如好友管理），我们打算使用自己定义的结构体以及自己用简单操作定义的函数来实现。对于客户端需要和服务器端共同进行的操作，我们打算使用 C 语言里面的网络编程函数以及相关的知识来实现，这一步是实现整个实时通讯系统的关键以及难点。

3.2 模块 2 ： 服务器端

文件名	对应模块	功能简介
im_server.h	无	包含所有模块的有文件
list .h/.c	链表管理	链表常用操作
msg_type.h	消息处理	消息类型及结构体定义
Serv_enc .h/c		服务器消息封装
Serv_dec .h/c		服务器消息解析
Serv_handle .h/c		服务器消息处理
Serv_user .h/c	用户管理	服务器用户管理
Serv_main .c	主程序	服务程序运行流程
Serv_config.h	常用配置	服务器常用配置头文件

3.2.1 功能描述

服务器端首先通过 socket 初始化，之后在用 bind 与固定的 ID 地址进行绑定，在绑定之后就进入 listen 的状态，listen 状态是不断地等待客户端请求的状态。

在 listen 状态里面，如果服务器在收到了客户端的请求，那么服务器就会进行接受（accept）的操作，即接受客户端的请求，然后发送给指定的另一个服务器信息。

如果在接受客户端的请求的时候有另外的一个客户端在发送请求，那么服务器端就会创建服务器子线程（一个小型的线程）来对另外的这一个客户端进行响应的操作。

3.2.2 接口描述

同样的，在实现服务器端的时候，我们也是先从 main()函数进入整个程序，main()函数是整个服务器端的入口，在使用整个 main()函数的时候，先进行参数的一个输入，在输入完所有的参数之后，我们需要进行服务器端的一个初始化，初始化失败会返回错误的提醒信息。初始化完成之后，整个程序会进入一个主循环之中去来等待与客户端的连接这一步会一直持续下去知道客户端发来了消息。

客户端在发来了消息之后，服务器端就会对消息进行接受，并且对消息进行一系列的操作（包括对消息的传递，错误报告等等）。在所有的进程结束之后，服务器端结束的时候，程序会对 socket 和 WSAS 等等进行清除与关闭。

主函数里面会涉及到所有的我们写到的函数与文件，比如用户管理相关的文件 Serv_user .h/c，大量的消息处理（msg_type.h，Serv_enc .h/c，Serv_dec .h/c，Serv_handle .h/c）等等。在上面所写到的所有对应的模块，在客户端发送过来相应的请求的时候都会在主函数里面进行调用以及实现。

3.2.3 数据结构描述

（1）服务器端同样使用了双向循环链表。在对好友进行增加，删除，查询的时候都需要大量使用到双向循环链表来进行实现。

（2）本模块同样使用了循环遍历来进行消息（包括账号和密码）的加密与解密，加密与解密的方式是遍历消息的每一个字符然后判断是否要加上 key 值，解密以相反的操作减去 key 值。

（3）本通讯系统的服务同样使用了很多有关 socket 的函数来实现网络编程，在实现该服务器的时候，服务器端需要先建立 socket，之后使用 bind()绑定 ip 地址等等。这里使用得到数据结构是有关网络编程的大量函数。

（4）以下是客户端与服务器端的交互消息类型，同样也是服务器端使用的消息数据结构。

编号	消息类型	消息子类型	简介
1	注册消息	无	用户注册向服务器发起注册
2	登录认证消息	无	用户登录认证消息
3	好友管理消息	好友状态查询	查询该用户是否在线
		好友列表请求	客户端服务器请求好友列表
		增加好友	客户端服务器请求增加好友
		删除好友	客户端服务器请求删除好友
		列出所有用户	客户端服务器请求列出所有用户
4	聊天消息	无	用户向其他用户发生聊天信息
5	退出登录	无	用户向服务器发起退出登录消息

3.2.4 实现思路

利用C语言原有的库以及从powershell窗口打开时候可以输入多个main()函数参数来进行对整个服务器端的实现，服务器端的所有的函数以及头文件都写在同一个目录下面，在打开服务器端的时候就在该文件目录下面进行操作即可。对于客户端需要和服务器端共同进行的操作，我们打算使用C语言里面的网络编程函数以及相关的知识来实现，这一步是实现整个实时通讯系统的关键，同样也是服务器端主要需要做到的功能。对于服务器端内部需要做到的操作（比如进行消息记录的保存等等），我们就使用简单的文件操作来进行文件的保存，我们会将文件保存到我们指定的本地目录里面去。

4.数据库与数据结构设计

4.1 数据结构设计

```
struct sockaddr_in
{
sa_family_t sin_family;//协议族 2字节
in_port_t sin_port;//端口号 2字节
struct in_addr sin_addr;//ip地址 4字节
char sin_zero[8]//填充，不起什么作用 8字节
};
struct sockaddr
{
sa_family_t sa_family; // 2字节 协议族
char sa_data[14] //14字节 字符数组
};
typedef struct client_friend
{
    //好友name
          char c_name[MAXNAME_LEN];
```

```
        //好友 id
        int c_id;
    //好友链表节点成员
        struct list_head c_node;
        //好友状态
        int c_stat;
}CLIENT_FRND;
typedef struct myself   //包含用户本身的信息的结构体
{
        /*user id*/
        int w_id;
        /*user name */
        char w_name[MAXNAME_LEN];
        /*user passwd*/
        char w_pass[MAX_USERPASS_LEN];
        /*user socket fd  用户的 socket 状态字*/
        int w_sockfd;
        /*user state : init, login  用户的状态*/
        int w_cstst;
        /*msg cnt 消息的数量*/
        int w_msgcnt;
        /*msg head 好友相关*/
        struct list_head w_msghd;
        /*all friends user list head 所有的好友的头结点*/
        struct list_head w_flisthd;
        /*friend cnt 好友的数量*/
        int w_fndcnt;
        //buffer 缓存区
        char w_buf[MAX_MSG_SIZE];
        /*message state : sending, recving  消息状态*/
        char w_mstat;
        /*expected msg tpye 消息类型*/
```

```
        char w_msgtype;
}MYSELF;
typedef struct usr_md
{
        char *u_str;
        short u_local;
        short u_mtype;
        int  u_optcnt;
}USR_CMD;
struct list_head //双向循环列表的前向指针和后向指针
{
        struct list_head  *next, *prev;
};
typedef struct msg_header/*实现通用消息结构，这是客户端和服务器端交互的消息的统一结构*/
{
        /*消息类型*/
        unsigned short msg_type;
        /*消息长度*/
        unsigned short msg_len;
        /*可变长度的消息内容,这是一个虚拟的消息内容指针(数组首地址的使用方法)，不占空间，仅仅是一个占位符*/
        char msg_data[0];
}MSG_HDR;//别名为 MSG_HDR
typedef  struct reg_msgdata//注册消息
{
        char r_name[MAXNAME_LEN];//昵称
        char r_passwd[MAX_USERPASS_LEN];//密码
}REG_MSG;//别名为 REG_MSG
typedef struct reg_msg_resp//注册后服务器回复的消息
{
        /*将 uid 返回给客户端，-1 表示出错*/
```

```
        int re_id;
        /*出错的原因，这个东西实际上是一个指针而非数组*/
        char re_reason[0];
}REG_RESP;
typedef struct login_msg//登录消息
{
        int lg_id;
        char lg_pass[MAX_USERPASS_LEN];
}LIG_MSG;
typedef struct login_resp//登陆回复消息，由服务器给客户端发送
{
        int lg_stat;/*登录状态 1 表示 ok  2 表示出错*/
        char lg_name[MAXNAME_LEN];//返回用户名
        char lg_reason[0];//返回错误原因
}LIG_RESP;
typedef struct frnd_op/*使用 msg_type 对好友进行操作*/
        {/*用户本身*/
        int f_id;
        /*表示需要操作的用户 id 数组*/
        int f_fids[0];
}FRND_OP;
typedef struct frnd_op_resp//对好友操作的回复
{
    //好友的在线状态
        short fre_stat;
        /*成功操作的数目*/
        short fre_num;
        /*好友的基本信息*/
        FRND_ST fre_ok_frnd[0];
}FRND_RESP;
typedef struct frnd_stat
{
```

```
        char fs_name[MAXNAME_LEN];
        int fs_id;
        int fs_stat;
}FRND_ST;
typedef struct chat_msg
{
        /*发送者 id*/
        int ch_sid;
        /*接收者 id*/
        int ch_rid;
        /*需要发送的文本信息*/
        char ch_msg[0];//这里使用的是服务器中转的方式进行发送消息
}CHAT_MSG;
typedef struct logout_msg//用户登出的确认(有点类似于登录，需要输入用户名和密码)
{
        int lg_id;
        char lg_pass[MAX_USERPASS_LEN];
}LOUT_MSG;
typedef struct im_user//用户结构体
{
        char u_name[MAXNAME_LEN];
        char u_pass[MAX_USERPASS_LEN];
        int u_id;
        int u_stat;//登陆状态
        int u_sckfd;//文件描述符

        struct list_head u_frndhd;//链表
        int u_frndcnt;//好友数量（与客户端类似）
}IM_USER;
```

4.2 数据存储设计

软件的早期版本使用在服务器端创建文本文件的方式对数据进行保存，分别创建 serv_user_file、serv_friend_file 以及 serv_chat_file 等文件。在使用文件对数据进行保存的过程中我们要使用到 C 中的文件读写模块。下面简单介绍一下其中一些常用函数的用法和参数的含义。

```
snprintf(char *str, size_t size, const char *format, ...)
    //将可变参数 “…” 按照 format 的格式格式化为字符串，然后再将其拷贝至 str 中。
    fgets()
    fputs()
    strtok_s( char *strToken, const char *strDelimit, char **buf)
    //当 strtok()在参数_strToKen 的字符串中发现参数*strDelimit 中包涵的分割字符时，则会将该字符改为 buf 字符
    atoi(const char *str)//把参数 str 所指向的字符串转换为一个整数
File *fp//声明 fp 是指针,用来指向 FILE 类型的对象
//fgetc 是 file get char 的缩写，意思是从指定的文件中读取一个字符。fgetc() 的用法为：
int fgetc (FILE *fp);
//fp 为文件指针。fgetc() 读取成功时返回读取到的字符，读取到文件末尾或读取失败时返回 EOF。
//fputc 是 file output char 的所以，意思是向指定的文件中写入一个字符。fputc() 的用法为:
int fputc ( int ch, FILE *fp );
/*ch 为要写入的字符，fp 为文件指针。fputc() 写入成功时返回写入的字符，失败时返回 EOF，返回值类型为 int 也是为了容纳这个负数*/
//fgets() 函数用来从指定的文件中读取一个字符串，并保存到字符数组中，它的用法为：
char *fgets ( char *str, int n, FILE *fp );
//str 为字符数组，n 为要读取的字符数目，fp 为文件指针。
//fputs() 函数用来向指定的文件写入一个字符串，它的用法为：
int fputs( char *str, FILE *fp );
//str 为要写入的字符串，fp 为文件指针。写入成功返回非负数，失败返回 EOF
```

5.接口设计

5.1 外部接口

该程序的使用的外部接口是软件的相关文件目录下的启动的 powershell 窗口（或通过 powershell 窗口将窗口命令行定为到我们的软件目录之下）。通过 powershell 窗口来对整个程序进行启动，对该程序的 powshell 窗口进行各种命令行的输入，我们就可以正常地使用该软件。同时在该外部窗口就能完全对该软件进行各种操作，不需要进入 IDE 对代码进行各种修改和 Debug。

例：

```
(base) PS E:\C++\项目集合\实时通信系统Server\bin\Debug> 实时通信系统Server.exe
serv: waitting for client's connection......
```

```
usage:E:\C++\项目集合\实时通信系统\bin\Debug\实时通信系统.exe <IPv4 server ip>
(base) PS E:\C++\项目集合\实时通信系统\bin\Debug> 实时通信系统.exe 127.0.0.1
im_client(unknown)#help
客户端聊天软件命令帮助(<>为必选参数):
reg <name> <passwd> <passwd> 通过用户名和密码注册用户。
login <uid> <password>  用户登录。
flist 列出所有的好友用户。
add <uid> 根据好友id添加好友。
del <uid> 根据好友id删除好友。
alist 查询所有的在线用户。
stat <uid> 查询好友状态。
talk <uid> <msg> 发送消息给好友。
exit 用户退出。
help 打印帮助信息。
mesg <from> <to> 显示从form用户到to用户的聊天记录。
debug 显示当前用户信息。
```

5.2 内部接口

在服务器端和客户端里面，内部接口是通过各个文件的相互关联加上主函数和各个函数互相嵌套来实现的，主函数对各个函数进行嵌套操作构成整个程序需要进行的部分，在打开整个程序的时候就只需要打开整个函数的，而其他文件以及其中函数就充当被调用的作用。

```
#pragma once
#ifndef IM_SERV_H
#define IM_SERV_H


#include <WinSock2.h>
#include <stdio.h>
#include <windows.h>
```

```
#include <Ws2tcpip.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#include <pthread.h>
#pragma comment(lib, "Ws2_32.lib")
#pragma comment(lib, "pthreadVC1.lib")

#include "list.h"
#include "msg_type.h"
#include "serv_config.h"
#include "serv_dec.h"
#include "serv_enc.h"
#include "serv_user.h"
#include "serv_handle.h"

typedef struct thread_args
{
	pthread_t t_tid;
	int t_sckfd;
	/*每个用户的 id 关联一个子线程*/
	int t_id;
}THREAD_ARGS;
#include "im_serv.h"

int main(int argc, char **argv)
{
	int lfd; //listrn fd;

	lfd = serv_init(argc, argv);
	if (lfd < 0)
```

```
        {
                fprintf(stderr, "serv: socket init error!\n");
                return -1;
        }

        serv_main_loop(lfd);

        serv_sock_clean(lfd);

        return 0;
}
```

6.其他设计

6.1 实现 TCP 并发服务器

并发的概念可以理解为：QQ 服务器只有一个，但是可以保证有多个客户端同时运行，服务器能与多个客户端同时进行通信，这就叫并发。要实现 C/S 模式下的实时通信系统就必然要实现 TCP 并发服务器。

TCP 原本不是并发服务器，TCP 服务器同一时间只能与一个客户端通信 （开多个客户端连接服务器只会显示第一个连接的客户端信息）——因为 TCP 服务器的代码中有两个阻塞函数，这两个阻塞函数不能交换位置，只能顺序执行，故导致无法实现并发（accept 理论是用来支持多个服务器与客户端进行通信，但这取决于代码的编写）。而在 UDP 中因为只有一个 recvfrom 在服务器端所以可以实现并发，不需要考虑其他阻塞函数。这里我们思考为什么不能用 UDP 来实现服务器呢？因为 UDP 提供的不可靠无连接服务常用于电子邮件或语音通话等对精度要求不是很高的场景，但是对于聊天软件来说，发送方和接收方收到的消息不一样是可能导致严重错误的(设想你和老师进行非常重要的有关分数的沟通时消息出现了错误）。

常见的实现 TCP 并发服务器有两种方式：

1. 使用多进程实现 TCP 并发服务器

```
int sockfd = socket()
bind()
listen()

while(1)//要保证一直运行就需要一直循环
{
    acceptfd = accept()
    pid = fork();//父进程执行 accept，保证一直在 accept 阻塞；子进程执行通信
    if(pid > 0)
    {//父进程什么也没有执行
    }
    else if(pid == 0)
    {//子进程一直在 while1 里面，服务器的每一个子进程就对应一个客户端
    while(1)
    {recv()/send()}
    }
}
```

2. 使用多线程实现 TCP 并发服务器

```
void *thread_fun(void *arg)
{//通信在子函数中执行
while(1)
{
recv() / send()
}
}
sockfd = socket()//创建套接字
bind()
listen()

while(1)
```

```
{
accept()//阻塞请求
    //只要有客户端连接上，则创建一个子线程与之通信
pthread_create(, , thread_fun, );
pthread_detach();//将线程设置为分离态
}
```

我们用简单的代码分别演示了这两种方式如何实现，最终在本次产品的开发过程中我们采用了多线程的模式，主要因为线程间通信会更加灵活、线程的创建和销毁成本比较低以及线程间的切换调度成本更加低。