```c
//////////////// client.h
#ifndef _CLIENT_H
#define _CLIENT_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sqlite3.h>
#include <time.h>
#include <fcntl.h>
#include <sys/stat.h>
enum{LOGIN = 1,LOGOUT,REGISTER,CHECKON,TALK,SENDFILE,RECVFILE,QUIT,UNKNOWN,HELP};
typedef struct sockaddr_in SA4;
typedef struct sockaddr SA;
int pcommand(void);
void phelp(void);
int psendcmd(int sfd);
int plogin(int sfd);
void plogout(void);
int pregister(int sfd);
int pcheckon(int sfd);
int ptalk(int sfd);
int ptalk(int sfd);
void* thread_send(void* psfd);
void* thread_recv(void* psfd);
int pfile_recv(int sfd,char* filepath,char* fromname,char* toname);
int pfile_send(int sfd,char* filepath,char* toname);
int pquit(int sfd);
int punknown(void);
int pconnect(char* ip);

#endif//_CLIENT_H


#include "client.h" /////////////////// climain.c

int main(int argc,char** argv){
/*
    if(argc != 2){
        printf("Usage: clnt <ip>\n");
        return -1;
    }
    int sfd = pconnect(argv[1]);
*/
    int sfd = pconnect("127.0.0.1");
    if(sfd == -1){
        printf("pconnect fails\n");
        return -1;
    }
    printf("successfully connected.\n");

    while(1){
        switch(pcommand()){
            case LOGIN:
                plogin(sfd);
                break;
            case LOGOUT:
                plogout();
                break;
            case REGISTER:
                pregister(sfd);
                break;
            case CHECKON:
                pcheckon(sfd);
                break;
            case TALK:
                ptalk(sfd);
                break;
            case QUIT:
                pquit(sfd);
                break;
            case HELP:
                phelp();
                break;
            case UNKNOWN:
                punknown();
                break;
            default:
                break;
        }
    }
    return 0;
}
```

```c

#include "client.h" ///////////////// client.c

char cmd[32] = {0};
int logstatus = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int ncond = 0;
FILE* pfile = NULL;
char myname[32] = {0};

int pcommand(void){

    while(1){
        printf("\ncommand:");
        fflush(stdin);
        fgets(cmd,20,stdin);//包含'\n'
        if(strchr(cmd,' ')){
            printf("space is not permitted in command.\n");
            continue;
        }
        if(strlen(cmd) == 0){
            printf("command can't be null.\n");
            continue;
        }
        break;
    }
    if(!strcmp(cmd,"help\n"))
        return HELP;
    else if(!strcmp(cmd,"login\n"))
        return LOGIN;
    else if(!strcmp(cmd,"logout\n"))
        return LOGOUT;
    else if(!strcmp(cmd,"register\n"))
        return REGISTER;
    else if(!strcmp(cmd,"online\n"))
        return CHECKON;
    else if(!strcmp(cmd,"talk\n"))
        return TALK;
    else if(!strcmp(cmd,"quit\n"))
        return QUIT;
    else
        return UNKNOWN;
```

```c
}

void phelp(void){
    printf("    login       set logstatus on;\n"
           "    logout      set logstatus off\n"
           "    register    register user ID;\n"
           "    online      check online list;\n"
           "    talk        enter talkroom;\n"
           "    quit        quit this client.\n");
}

int psendcmd(int sfd){

    strtok(cmd,"\n");
    dprintf(sfd,"%s\n",cmd);
//  printf("command sent: %s\n",cmd);
    return 0;
}

//注意:logstatus 是存储在客户端本地的登录状态判断条件，
//其状态是其他有关功能开启或终止的前提条件，
//但服务器仅将登入 chatroom 的用户计入 online list，
//以便更有针对性的统计处于聊天状态的实时有效用户。
//plogin 会通过服务器进行对比验证，但服务器不存储其登录状态
//plogin 成功后，logstatus 置为 1，否则 logstatus 等于初值 0
//plogout 将 logstatus 重新置为 0。

void plogout(void){
    if(logstatus == 1)
        logstatus = 0;

    printf("logstatus off!\n");
}

int plogin(int sfd){

    if(logstatus == 1){
        printf("can not relogin, please retry.\n");
        return -1;
    }

    if(psendcmd(sfd) == -1)
        return -1;

    char buf[100] = {0};
    char username[32],password[32];
    while(1){
        printf("username:");
        fgets(username,32,stdin);//包含'\n'
        if(strchr(username,' ')){
            printf("space is not permitted in username.\n");
            continue;
        }
        if(strlen(username) > 24){
            printf("username should be <= 24 characters\n");
            continue;
        }
        break;
    }

    while(1){
        strcpy(password,getpass("password:"));//包含'\n'
        if(strchr(password,' ')){
            printf("space is not permitted in password.\n");
            continue;
        }
        if(strlen(password) > 24){
            printf("password should be <= 24 characters.\n");
            continue;
        }
        break;
    }
    strtok(username,"\n");
    strtok(password,"\n");
    dprintf(sfd,"%s %s\n",username,password);
    //全局变量 myname 赋值
    strcpy(myname,username);

    int n = 0;
    if((n = read(sfd,buf,100)) < 0 ){
        printf("failed to read login reply from server.\n");
        return -1;
    }
    buf[n] = '\0';

    if(strstr(buf,"successful")){
        logstatus = 1;
        printf("logstatus on!\n");
    }else
        printf("%s",buf);

    return 0;
}

int pregister(int sfd){
    if(logstatus == 1){
        printf("please logout first!.\n");
        return -1;
    }

    if(psendcmd(sfd) == -1)
        return -1;

    char buf[100] = {0};
    char username[32],password[32];
    while(1){
        printf("username:");
        fgets(username,32,stdin);//包含'\n'
        if(strchr(username,' ')){
            printf("space is not permitted in username\n");
            continue;
        }
        if(strlen(username) > 24){
            printf("username should be <= 24 characters\n");
            continue;
        }
        if(!strcmp(username,".")){
            printf("username should not be a '.' \n");
            continue;
        }
        break;
    }

    while(1){
        strcpy(password,getpass("password:"));//包含'\n'
        if(strchr(password,' ')){
            printf("space is not permitted in password.\n");
            continue;
        }
        if(strlen(password) > 24){
            printf("password should be <= 24 characters.\n");
            continue;
        }
        break;
    }
    char tmppass[32] = {0};
    while(1){
        strcpy(tmppass,getpass("confirm password:"));//包含'\n'
        if(strchr(tmppass,' ')){
            printf("space is not permitted in password.\n");
            continue;
        }
        if(strlen(tmppass) > 24){
            printf("password should be <= 24 characters.\n");
            continue;
        }
        break;
    }

    if(strcmp(password,tmppass)){
        printf("password inputs differ,pleaes re_register.\n");
        dprintf(sfd,"register failed\n");
        return -1;
    }

    strtok(username,"\n");
    strtok(password,"\n");
    dprintf(sfd,"%s %s\n",username,password);

    int n = 0;
    if((n = read(sfd,buf,100)) < 0 ){
        printf("failed to read register reply from server.\n");
        return -1;
    }
    buf[n] = '\0';
    printf("%s",buf);

    return 0;
}

int pcheckon(int sfd){

    if(logstatus == 0){
        printf("please login first!\n");
        return -1;
    }
    if(psendcmd(sfd) == -1)
        return -1;

    int cnt = 0;
    char buf[16] = {0};
    int n = 0;
    if((n = read(sfd,buf,16)) <= 0)
        printf("failed to get size of userlist.\n");
    buf[n] = '\0';
    sscanf(buf,"%d\n",&cnt);
    printf("members online: %d\n",cnt);
    if(cnt == 0) return 0;

    char* userlist = (char*)malloc(32*cnt+100);//彻底杜绝内存不够?32 不是已经够了吗
    if(userlist == NULL){
        printf("mem error:failed to malloc mem for userlist.\n");
        return -1;
    }
    userlist[0] = '\0';
    //printf("malloc done. starting reading userlist.\n");
    //为什么服务器发送成功了,但是 read()函数经常不返回?
    if((n = read(sfd,userlist,32*cnt+100)) <= 0){
        printf("failed to get userlist from server.\n");
        return -1;
    }
    userlist[n] = '\0';//如果内存不足的话,有可能设置字符串结尾\0 失败
    //字符串\0 结尾设置不成功的话,就会无法正常输出
    printf("%s\n",strtok(userlist,"\n"));//接收到的 userlist 自带\n
    free(userlist);
    userlist = NULL;

    return cnt;
}

int ptalk(int sfd){

    if(logstatus == 0){
        printf("please login first!\n");
        return -1;
    }
    if(psendcmd(sfd) == -1)
        return -1;

    char reply[128] = {0};
```

```c
    int r = 0;
    if((r = read(sfd,reply,128)) < 0){
        printf("failed to get reply from server!\n");
        return -1;
    }
    reply[r] = '\0';
    if(!strstr(reply,"successful")){
        printf("%s\n",reply);
        return -1;
    }

    time_t t = time(NULL);
    struct tm *today = localtime(&t);
    char date[32] = {0};
    sprintf(date,"%02d%02d%02d",today->tm_year+1900,today->tm_mon+1,today->tm_mday);

    char logname[256] = {0};
    strcpy(logname,myname);
    strcat(logname,"_chatlog_");
    strcat(logname,date);
    strcat(logname,".txt");

    pfile = fopen(logname,"a");
    if(pfile == NULL)
        printf("failed to open chatlog.\n");

    printf("\n");
    pthread_t tid1,tid2;
    if(pthread_create(&tid1,0,thread_send,(void*)&sfd) != 0){
        dprintf(sfd,":exit\n");
        printf("error: failed to create thread_send.\n");
        return -1;
    }
    if(pthread_create(&tid2,0,thread_recv,(void*)&sfd) != 0){
        dprintf(sfd,":exit\n");
        printf("error : failed to create thread_recv.\n");
        return -1;
    }

    if(pthread_join(tid1,NULL) == 0 || pthread_join(tid2,NULL) == 0){
        pthread_cancel(tid1);
        pthread_cancel(tid2);
    }

    fclose(pfile);
    pfile = NULL;
    return 0;
}

void* thread_send(void* psfd){

    time_t t = 0;
    struct tm *today = NULL;
    int sfd = *(int*)psfd;
    char msg[1000] = {0};
    char filepath[100] = {0};
    char toname[32] = {0};
    char atme[32] = {0};
    strcat(atme,"@");
    strcat(atme,myname);

    while(1){
        fgets(msg,1000,stdin);//包含\n\0

        if(strstr(msg,atme))
            continue;
        if(!strcmp(msg,"\n"))//空白消息,只包含\n 字符
            continue;

        //规定群发@.之后，所有消息都带有@
        if(msg[0] == '@'){//如果指定接收人，则修改 toname 为给定值;

            if(strstr(msg,":file")){
                sscanf(msg,"@%s",toname);
                if(strlen(toname) == 1 && toname[0] == '.'){
                    printf("can not broadcast file by @.\n");
                    continue;
                }

                if(!strstr(msg,"$")){
                    printf("$filepath should be designated.\n");
                    continue;
                }
                sscanf(msg,"%*[^$]$%s",filepath);

                dprintf(sfd,"%s",msg);//msg 包含@toname 和\n
                //首先判断 toname 是否存在,如果不存在,则返回
                pthread_mutex_lock(&mutex1);
                pthread_cond_wait(&cond,&mutex1);//经过通知,才能开始发送
                pthread_mutex_unlock(&mutex1);
                if(ncond != 1) continue;
                if(pfile_send(sfd,filepath,toname) == -1) continue;
                ncond = 0;
            }else
                dprintf(sfd,"%s",msg);//msg 包含@toname 和\n
        }else{//群发,补加@.
            if(strstr(msg,":file")){//不允许进行文件群发
                printf("@toname should be designated.\n");
                continue;
            }
            dprintf(sfd,"@. %s",msg);//msg 包含\n
        }

        t = time(NULL);
        today = localtime(&t);
        pthread_mutex_lock(&mutex);
        fprintf(pfile,"%02d:%02d:%02d %s\n",today->tm_hour,today->tm_min,today->tm_sec,msg);
        pthread_mutex_unlock(&mutex);
        if(!strcmp(msg,":exit\n"))
            return (void*)0;
    }
}

void* thread_recv(void* psfd){

    time_t t = 0;
    struct tm *today = NULL;
    int sfd = *(int*)psfd;
    char msgbuf[1000] = {0};
    char realmsg[1000] = {0};
    char filepath[100] = {0};
    char fromname[32] = {0};
    char toname[32] = {0};
    int lenfrom = 0;
    int lento = 0;

    int n = 0;
    while(1){//服务器转发不再对字符串进行任何处理,如果原来包含\n,那么现在仍然有\n
        if((n = read(sfd,msgbuf,1000)) <= 0){//若服务器退出, 则退出
            perror("read");
            return (void*)-1;
        }
        msgbuf[n] = '\0';
        //所有的消息格式都是 msgbuf = fromname:@toname realmsg
        sscanf(msgbuf,"%[^:]",fromname);//:之前的所有字符
        lenfrom = strlen(fromname);
        sscanf(msgbuf,"%*[^@]@%s",toname);
        lento = strlen(toname);
        strcpy(realmsg,msgbuf+lenfrom+lento+3);
    //printf("fromname=%s toname=%s realmsg=%s",fromname,toname,realmsg);

        //若对方确认接受文件,则设置 ncond 值
        if(!strcmp(realmsg,"[verify]: OK.\n")){
            ncond = 1;
            pthread_cond_signal(&cond);
        }
        if(!strcmp(realmsg,"[verify]: NO.\n")){
            ncond = 0;
            pthread_cond_signal(&cond);
        }
        if(!strcmp(realmsg,"[verify]: CC.\n")){
            ncond = 2;
            pthread_cond_signal(&cond);
        }
        if(!strcmp(realmsg,"[verify]: SS.\n")){
            ncond = -1;
            pthread_cond_signal(&cond);
        }
        if(!strcmp(realmsg,"@toname not online!\n")){
            ncond = -1;
            pthread_cond_signal(&cond);
        }

        if(!strstr(realmsg,"[verify]:")){//不显示[verify]:消息
            //群发则不含@toname，realmsg 包含\n
            if(strlen(toname) == 1 && toname[0] == '.')
                printf("%s:%s",fromname,realmsg);
            else
                printf("%s:@%s %s",fromname,toname,realmsg);
        }

        //此时 msg 不包含 fromname:@toname
        if(strstr(realmsg,":file") && strstr(realmsg,"$")){
            sscanf(realmsg,"%*[^$]$%s",filepath);
            if(pfile_recv(sfd,filepath,fromname,toname) == -1){
                continue;//文件接收失败的话，接收请求就不写入日志
            }
        }

        t = time(NULL);
        today = localtime(&t);
        pthread_mutex_lock(&mutex);
        fprintf(pfile,"%02d:%02d:%02d %s\n",today->tm_hour,today->tm_min,today->tm_sec,realmsg);
        pthread_mutex_unlock(&mutex);
    }

    return (void*)-1;
}

int pfile_send(int sfd,char* filepath,char* toname){
    printf("pfile_send: start sending..\n");

    //toname 最长 25 个字节
    //解析文件名
    char path[100] = {0};
    char childpath[100] = {0};
    char* name = NULL;
    char cwd[100] = {0};
    char tmpcwd[100] = {0};
    char* curwd = NULL;
    getcwd(cwd,100);
    getcwd(tmpcwd,100);
    int len = strlen(toname);

    //解析目标路径
    if(strstr(filepath,"/")){
        name = 1 + strrchr(filepath,'/');
        strcpy(childpath,filepath);
        //将 childpath 倒数第一个/ 设置为\0
        strrchr(childpath,'/')[0] = '\0';
    }else{
        name = filepath;
        strcpy(childpath,cwd);
    }
    printf("path=%s name=%s\n",childpath,name);

    //解析真实路径
    //1 ~ home 目录起头
    if(childpath[0] == '~'){
        //1.1 有子目录
        if(strlen(childpath) >1){
            strcpy(path,getenv("HOME"));
            strcat(path,strtok(childpath,"~"));
            //strtok()一般情况下,将出现的字符全部设置为\0,
            //然后返回剩下的字符串中不为\0 的首地址
        }else
            //1.2 没有子目录
            strcpy(path,getenv("HOME"));
    //2 / 根目录起头
    }else if(childpath[0] == '/')
        strcpy(path,childpath);
    //3 .. 上层目录起头
    else if(strlen(childpath) > 1 && childpath[0] == '.' && childpath[1] == '.'){
        strcpy(path,cwd);//拷贝当前目录
        //将倒数第一个/ 设置为\0 , 所得即是上层目录
        strrchr(path,'/')[0] = '\0';
        //3.1 有子目录
        if(strlen(childpath) > 2)
            //直接跳过.. 将后面的子目录连缀至上层路径 path
            strcat(path,childpath+2);
        //3.2 没有子目录
            //什么都不干
    //4 . 当前目录起头
    }else if(childpath[0] == '.'){
        //path 保存当前路径
        strcpy(path,cwd);
        //4.1 有子目录
        if(strlen(childpath) > 1)
            //跳过. 并连缀到当前目录
            strcat(path,childpath+1);
        //4.2 没有子目录
            //啥都不干
    //5 其他任意字符起头 通常表示当前目录下的子目录
    }else{
        strcpy(path,cwd);
        strcat(path,"/");
        strcat(path,childpath);
    }
    //路径解析完成
    chdir(path);
//  printf("working directory changed as:%s\n",path);

```

```
    //获取文件大小
    int size = 0;
    struct stat filestat = {0};
    if(stat(name,&filestat) == -1){
        dprintf(sfd,"@%s $staterr$\n",toname);
        perror("stat error");
        printf("\n");
        return -1;
    }
    size = filestat.st_size;

    if(size == 0){
        dprintf(sfd,"@%s $sizeerr$\n",toname);
        printf("filesize=0,failed to send file.\n\n");
        return -1;
    }
    //$file$在服务器转发过程中有特殊意义,
    //表示以原字符串风格转发,不添加来源姓名
    dprintf(sfd,"@%s filesize=%d\n",toname,size);

    //注意,由于消息接收线程的持续存在,消息发送线程实际是收不到认证消息的
    //所以需要通过 cond 条件变量,实现收发线程间的同步
    pthread_mutex_lock(&mutex1);
    pthread_cond_wait(&cond,&mutex1);
    pthread_mutex_unlock(&mutex1);
    if(ncond != 1){
        printf("error: recver failed to recv file.\n");
        return -1;
    }

    FILE* psendfile = fopen(name,"r");
    if(psendfile == NULL){
        dprintf(sfd,"@%s $openerr$\n",toname);
        perror("fopen error");
        printf("\n");
        return -1;
    }

    int n = 0,w = 0;
    int wsum = 0;
    char filebuf[900] = {0};//不超过服务器接收范围
    while(1){
        pthread_mutex_lock(&mutex1);
        pthread_cond_wait(&cond,&mutex1);//经过信号量通知,才能开始发送
```

```c
            pthread_mutex_unlock(&mutex1);
            if(ncond != 2){
                printf("error:file sending process failed.\n");
                return -1;
            }

            if((n = fread(filebuf,1,900,psendfile)) < 0){
                ferror(psendfile);
                return -1;
            }
            filebuf[n] = '\0';

            //如果不指定 toname,则 toname = ".";
            w = dprintf(sfd,"@%s %s\n",toname,filebuf);//增加\n 以出尽缓存

            wsum += w-len-3;
            printf("sent: %d bytes, %%%.2lf...\n",w-len-3,wsum*100.0/size);
            ncond = 1;
            if(wsum >= size)  break;
        }

        ncond = 0;//发送完毕之后重置判断条件
        fclose(psendfile);
        psendfile = NULL;
        chdir(cwd);
        printf("file size=%d sent successful.\n\n",size);

        return 0;
    }

    int pfile_recv(int sfd,char* filepath,char* fromname,char* toname){
        char* name = NULL;
        if(strstr(filepath,"/"))
            name = 1 + strrchr(filepath,'/');
        else
            name = filepath;
        printf("name=%s\n",name);

        //因为不允许进行文件群发,所以所有的文件转发都是定向单发

        //获取文件大小
        int size = 0;
        char sizebuf[64] = {0};
        int n = 0;
        if((n = read(sfd,sizebuf,32)) < 0){
            perror("read error");
            printf("\n");
            return -1;
        }
        sizebuf[n] = '\0';
        if(strstr(sizebuf,"$staterr$") || strstr(sizebuf,"$sizeerr$")){
            printf("sender failed to fetch file size.\n\n");
            return -1;
        }
        sscanf(sizebuf,"%*s filesize=%d\n",&size);
    //  printf("size=%d\n",size);

        //根据文件大小,选择发送不同的认证消息

        if(size == 0){
            dprintf(sfd,"@%s [verify]: NO.\n",fromname);
            printf("filesize=0,failed to create file.\n\n");
            return -1;
        }

        dprintf(sfd,"@%s [verify]: OK.\n",fromname);

        FILE* precvfile = fopen(name,"w");
        if(precvfile == NULL){
            dprintf(sfd,"@%s [verify]: SS.\n",fromname);
            perror("fopen error");
            printf("\n");
            return -1;
        }

        int lenfrom = strlen(fromname);
        int lento = strlen(toname);
        int r = 0;
        int w = 0;
        int wsum = 0;
        char filebuf[1000] = {0};
        char realmsg[1000] = {0};
        int lenreal = 0;
        //因为 read()返回次数不确定，所以循环次数不可以与发送次数一致
        while(1){
            //通知发送方可以发送了
            dprintf(sfd,"@%s [verify]: CC.\n",fromname);
            r = read(sfd,filebuf,1000);//首先进入等待状态,阻塞接收
            if(r < 0){//格式为 fromname:@toname realmsg\n
                dprintf(sfd,"@%s [verify]: SS.\n",fromname);
                perror("read error");
                printf("\n");
                ferror(precvfile);
                return -1;
            }
            filebuf[r] = '\0';
            //一共接收 r 个有效字符,
            //格式为 fromname:@toname realmsg\n
            strcpy(realmsg,filebuf+lenfrom+lento+3);
            //绝对不能用 sscanf(),因为它遇空格或者换行就会停止

            if(strstr(realmsg,"$openerr$") || strstr(realmsg,"$readerr$")){
                dprintf(sfd,"@%s [verify]: SS.\n",fromname);
                printf("sender failed to send file content.\n\n");
                return -1;
            }
            lenreal = strlen(realmsg);//包含\n
            realmsg[lenreal-1] = '\0';// \n 替换为\0

            if((w = fwrite(realmsg,1,strlen(realmsg),precvfile)) < 0){
                dprintf(sfd,"@%s [verify]: SS.\n",fromname);
                ferror(precvfile);
                return -1;
            }

            wsum += w;
            printf("recved: %d bytes, %%%.2lf...\n",w,wsum*100.0/size);
            if(wsum >= size) break;
        }

        fclose(precvfile);
        precvfile = NULL;
        printf("file size=%d recved successful.\n\n",size);

        return 0;
    }

    int pquit(int sfd){

        psendcmd(sfd);
        exit(0);
    }
```

```c

int punknown(void){

    printf("command is not known,please reinput.\n");
    return 0;
}


int pconnect(char* ip){

    SA4 serv;
    serv.sin_family = AF_INET;
    serv.sin_port = htons(8080);
    serv.sin_addr.s_addr = inet_addr(ip);

    int sfd = socket(AF_INET,SOCK_STREAM,0);
    if(sfd == -1){
        perror("socket");
        return -1;
    }

    int c = connect(sfd,(SA*)&serv,sizeof(serv));
    if(c == -1){
        perror("connect");
        return -1;
    }
    return sfd;
}

/////////////////////// server.h
#ifndef _SERVER_H
#define _SERVER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <sqlite3.h>
enum{LOGIN = 1,REGISTER,CHECKON,TALK,SENDFILE,QUIT};
enum{SQL_ERROR = -1,SQL_NONE,SQL_FOUND};

```

```c
typedef struct sockaddr_in SA4;
typedef struct sockaddr SA;

typedef struct node{
    char username[32];
    int tcfd;
    struct node* pprev;
    struct node* pnext;
}node;

typedef struct list{
    node* pcur;
    node head;
    node tail;
}list;

void* pexit(void*);
void* pnewthread(void* pcfd);
int pcommand(int cfd);
int plogin(int cfd,char** pmyname);
int pregister(int cfd);
int pcheckon(int cfd);
int ptalk_transfer(int cfd,char* myname);
void pgroupmsg(int mycfd,char* myname,char* msg);
int pquit(int cfd);
int plisten(int port,int backlog);

int list_init(list* plist);
int list_count(list* plist);
int list_show(list* plist,int cfd);
int list_getcfd(const char* username,list* plist);
int* list_getcfdarr(int** pcfdarr,int* pcnt,list* plist);
char* list_getname(int cfd,list* plist);
int list_append(const char* username,int cfd,list* plist);
int list_delete(int cfd,list* plist);
int list_destroy(list* plist);

int db_open(const char* dbname,sqlite3* pdb);
int db_check(const char* username,const char* password,const char* dbname,sqlite3*
pdb);
int db_insert(const char* username,const char* password,const char* dbname,sqlite3*
pdb);
int db_delete(const char* username,const char* dbname,sqlite3* pdb);
static int callback(void* data,int argc,char** argv,char** azcolname);
```

```

#endif//_SERVER_H

#include "server.h"////////////////// servermain.c

list users;
sqlite3* pdb;
const char* dbname = "chat.db";

int main(int argc,char** argv){
    SA4 client;
    socklen_t clilen = sizeof(client);

    int sfd = plisten(8080,6);
    if(sfd == -1){
        printf("plisten failed.\n");
        return -1;
    }
    printf("start listening ...\n");
    //初始化在线用户链表
    list_init(&users);
    //创建数据库 并创建用户注册表
    if(db_open(dbname,pdb) == -1)
        return -1;

    pthread_t tid0;
    int ret = pthread_create(&tid0,0,pexit,NULL);
    if(ret != 0){
        printf("error %d: pthread_create failed.\n",ret);
        return -1;
    }

    while(1){
        char IP[32] = {0};
        int cfd = accept(sfd,(SA*)&client,&clilen);
        if(cfd == -1){
            perror("accept");
            return -1;
        }

        pthread_t tid;
        int t = pthread_create(&tid,0,pnewthread,(void*)&cfd);
        if(t != 0){
            printf("error %d: pthread_create failed.\n",t);
```

```c
                return -1;
            }
            printf(/*"%s: */"client thread cfd=%d 
created.\n",/*inet_ntop(AF_INET,&client.sin_addr,IP,32),*/cfd);
        }

        return 0;
}


#include "server.h"////////////////// server.c

extern list users;
extern sqlite3* pdb;
extern const char* dbname;

void* pexit(void* null){
    char cmd[32] = {0};
    while(1){
        fgets(cmd,32,stdin);//fgets()获取的字符串包含\n
        if(!strcmp(cmd,":exit\n"))
            exit(0);
    }
}

void* pnewthread(void* pcfd){

    char* myname = NULL;
    int cfd = *(int*)pcfd;

    while(1){
        switch(pcommand(cfd)){
            case LOGIN:
                plogin(cfd,&myname);
                break;
            case REGISTER:
                pregister(cfd);
                break;
            case CHECKON:
                pcheckon(cfd);
                break;
            case TALK:
                ptalk_transfer(cfd,myname);
                break;
            case QUIT:
                pquit(cfd);
                break;
            default:
                break;
        }
    }
    return (void*)0;
}

int pcommand(int cfd){

    char cmd[32] = {0};
    int n = 0;
    if((n = read(cfd,cmd,32)) < 0){
        perror("read error");
        return QUIT;//如果读不到command,就会发出退出命令
    }
    cmd[n] = '\0';

    if(!strcmp(cmd,"login\n"))
        return LOGIN;
    else if(!strcmp(cmd,"register\n"))
        return REGISTER;
    else if(!strcmp(cmd,"online\n"))
        return CHECKON;
    else if(!strcmp(cmd,"talk\n"))
        return TALK;
    else if(!strcmp(cmd,"sendfile\n"))
        return SENDFILE;
    else if(!strcmp(cmd,"quit\n"))
            return QUIT;

    return QUIT;
}

int plogin(int cfd,char** pmyname){

    char buf[100] = {0};
    char username[32] = {0},password[32] = {0};

    int n = 0;
    if((n = read(cfd,buf,100)) < 0){
        printf("failed to read login message from client.\n");
        return -1;
    }
    buf[n] = '\0';
    sscanf(buf,"%s %s\n",username,password);

    switch(db_check(username,password,dbname,pdb)){
        case SQL_NONE:
            dprintf(cfd,"username or password wrong!\n");
//          printf("username or password wrong!\n");
            break;
        case SQL_FOUND:
            dprintf(cfd,"login successful!\n");
            *pmyname = (char*)malloc(32);
            strcpy(*pmyname,username);
//          printf("%s cfd=%d login successful!\n",username,cfd);
            break;
        case SQL_ERROR:
            dprintf(cfd,"database currently unavailable,please retry later!\n");
            break;
        default:
            break;
    }
    return 0;
}

int pregister(int cfd){

    char buf[100] = {0};
    char username[32] = {0},password[32] = {0};

    int n = 0;
    if((n = read(cfd,buf,100)) < 0){
        printf("failed to read register message from client.\n");
        return -1;
    }
    buf[n] = '\0';
    sscanf(buf,"%s %s\n",username,password);
    if(!strcmp(username,"register") && !strcmp(password,"failed")){
//      printf("password inputs differ,client may retry.\n");
        return -1;
    }

    switch(db_check(username,password,dbname,pdb)){
        case SQL_NONE...SQL_FOUND:
```

```c
            if(db_insert(username,password,dbname,pdb) == 0){
                if(db_check(username,password,dbname,pdb) == SQL_FOUND){
                    dprintf(cfd,"user registered successfully!\n");
//                  printf("user registered successfully!\n");
                }
            }else{
                dprintf(cfd,"username already exists,please re_register!\n");
//              printf("username already exists,please re_register!\n");
            }
            break;
        case SQL_ERROR:
            dprintf(cfd,"database currently unavailable,please retry later!\n");
            break;
        default:
            break;
    }
    return 0;
}

int pcheckon(int cfd){

    list_show(&users,cfd);
    return 0;
}

int ptalk_transfer(int cfd,char* myname){

    if(list_getcfd(myname,&users) > 0){
        dprintf(cfd,"relogin: user is online somewhere else!\n");
        printf("relogin: user is online somewhere else.\n");
        return -1;
    }else{
        dprintf(cfd,"enter talkroom successful.");
//      printf("%d entered talkroom successful.\n",cfd);
    }

    char msg[1000] = {0};
    int tcfd = 0;
    char toname[32] = {0};
    list_append(myname,cfd,&users);

    int n = 0;
    int len = 0;
    while((n = read(cfd,msg,1000)) > 0){
        msg[n] = '\0';
        //msg 自带\n,尤其是文件内容,不能删掉
        if(!strcmp(msg,"@. :exit\n")){
            char exitmsg[100] = {0};
            sprintf(exitmsg,"@. [msg]:left talk.\n");
            pgroupmsg(cfd,myname,exitmsg);
            list_delete(cfd,&users);
            break;
        }

        //所有消息格式都为@toname realmsg\n
        //组织成新的格式为 fromname:@toname realmsg\n
        sscanf(msg,"@%s ",toname);
        len = strlen(toname);

        if(len == 1 && toname[0]=='.')//群发
            pgroupmsg(cfd,myname,msg);//包含@toname
        else{//单发
            if((tcfd = list_getcfd(toname,&users)) == -1){
                dprintf(cfd,"server:@%s @toname not online!\n",myname);
                continue;
            }
            if(strstr(msg,":file") && strstr(msg,"$"))//只对文件命令发送确认消息
                dprintf(cfd,"server:@%s [verify]: OK.\n",myname);
            dprintf(tcfd,"%s:%s",myname,msg);//包含@toname
    //      printf("transfer realmsg len=%lu.\n",strlen(msg)-len-2);
        }
    }
    //printf("ptalk_transfer exited.\n");
    return 0;
}
void pgroupmsg(int mycfd,char* myname,char* msg){

    int clients = 0;
    int* cfdarr = NULL;
    char toname[32] = {0};
    sscanf(msg,"@%s ",toname);
    int len = strlen(toname);

    //该函数会调用 malloc 所以用完之后 一定要 free
    list_getcfdarr(&cfdarr,&clients,&users);

    for(int i=0; i<clients; i++)
        if(cfdarr[i] != mycfd){
                dprintf(cfdarr[i],"%s:%s",myname,msg);//包含@toname
    //      printf("broadcast realmsg len=%lu.\n",strlen(msg)-len-2);
        }

    //free 临时数组的内存
    free(cfdarr);
    cfdarr = NULL;
}

int pquit(int cfd){

    if(list_getname(cfd,&users))//如果 cfd 所对应的用户存在,则删除之
        list_delete(cfd,&users);//针对意外退出情况
    printf("client thread cfd=%d exited.\n",cfd);
    pthread_exit(NULL);
}

int plisten(int port,int backlog){

    SA4 serv;
    serv.sin_family = AF_INET;
    serv.sin_port = htons(port);
    serv.sin_addr.s_addr = htonl(INADDR_ANY);

    int sfd = socket(AF_INET,SOCK_STREAM,0);
    if(sfd == -1){
        perror("socket");
        return -1;
    }

    int b = bind(sfd,(SA*)&serv,sizeof(serv));
    if(b == -1){
        perror("bind");
        return -1;
    }

    int l = listen(sfd,backlog);
    if(l == -1){
        perror("listen");
        return -1;
    }

    return sfd;
}
```

```c


#include "server.h" ///////////////////// list.c

int list_init(list* plist){

    plist->pcur = NULL;
    plist->head.pprev = NULL;
    plist->tail.pnext = NULL;
    plist->head.pnext = &plist->tail;
    plist->tail.pprev = &plist->head;

    printf("list_init successful.\n");
    return 0;
}

int list_count(list* plist){

    int cnt = 0;
    node* pnode = NULL;
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
        if(pnode != &plist->tail)
            cnt++;

    return cnt;
}

int list_show(list* plist,int cfd){

    int cnt = list_count(plist);
    dprintf(cfd,"%d\n",cnt);
    if(cnt == 0) return 0;

    char* userlist= (char*)malloc(32*cnt+100);//彻底杜绝内存不足? 32 不是已经够了吗?
    if(userlist == NULL){
        dprintf(cfd,"failed to get userlist.\n");
        printf("failed to get userlist.\n");
        return -1;
    }

    //使用 strcat()之前，一定要 bzero.bzero 这个函数貌似经常出错
    userlist[0] = '\0';

    node* pnode = NULL;
    int lensum = 0;
    //如果 plist->head.pnext == &plist->tail,即 plist 当中没有有效成员的话,就不会进行循环
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext){
        strcat(userlist,pnode->username);
        strcat(userlist," ");
        lensum += strlen(pnode->username)+1;
    }
    userlist[lensum-1] = '\0';
    dprintf(cfd,"%s\n",userlist);//userlist 发出去之后包含\n
    //printf("userlist sent:%s\n",userlist);
    free(userlist);
    userlist = NULL;
    return 0;
}

int list_getcfd(const char* username,list* plist){

    node* pnode = NULL;
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
        if(!strcmp(username,pnode->username))
            return pnode->tcfd;

    return -1;
}

int* list_getcfdarr(int** pcfdarr,int* pcnt,list* plist){
    *pcnt = list_count(plist);
    *pcfdarr = (int*)malloc(sizeof(int) * (*pcnt));
    if(*pcfdarr == NULL){
        printf("failed to malloc mem to init cfdarr[clients].\n");
        return NULL;
    }
    int i = 0;
    node* pnode = NULL;
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
        (*pcfdarr)[i++] = pnode->tcfd;

    return *pcfdarr;
}

char* list_getname(int cfd,list* plist){

    node* pnode = NULL;
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
        if(cfd == pnode->tcfd)
            return pnode->username;

    printf("failed to get name from list where cfd=%d\n",cfd);
    return NULL;
}

int list_append(const char* username,int cfd,list* plist){

    node* pnode = (node*)malloc(sizeof(node));
    if(pnode == NULL){
        printf("\nfailed to append %s into list.\n",username);
        return -1;
    }

    strcpy(pnode->username,username);
    pnode->tcfd = cfd;
    plist->tail.pprev->pnext = pnode;
    pnode->pprev = plist->tail.pprev;
    pnode->pnext = &plist->tail;
    plist->tail.pprev = pnode;
//  printf("user cfd=%d appended to list successful.\n",cfd);

    return 0;
}

int list_delete(int cfd,list* plist){

    node* pnode = NULL;
    for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext){
        if(cfd == pnode->tcfd){
            pnode->pprev->pnext = pnode->pnext;
            pnode->pnext->pprev = pnode->pprev;
            free(pnode);
            pnode = NULL;
//          printf("\nuser cfd=%d deleted from list successful.\n",cfd);
            return 0;
        }
    }
    printf("user cfd=%d does not exist!\n",cfd);
    return -1;
}
```

```c
int list_destroy(list* plist){

    plist->pcur = NULL;
    while(plist->head.pnext != &plist->tail){
        node* pfirst = &plist->head;
        node* pmid = pfirst->pnext;
        node* plast = pmid->pnext;

        pfirst->pnext = plast;
        plast->pprev = pfirst;
        free(pmid);
        pmid = NULL;
    }

    return 0;
}

#include "server.h" /////////////////// sqlitedb.c

int db_open(const char* dbname,sqlite3* pdb){
    char* sql = NULL;
    char* zerrmsg = NULL;

    if(sqlite3_open(dbname,&pdb) != 0){
        printf("database can not be opened.\n");
        return -1;
    }

    //sql 主键只能有一个
    sql = "create table chaters(\n"
        "username varchar(36) primary key not null,\n"
        "password varchar(36) not null);";

    int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
    if(rc != SQLITE_OK){
        sqlite3_close(pdb);
        printf("sql: %s\n",zerrmsg);
        sqlite3_free(zerrmsg);
        return 0;//数据表已经存在而导致键表不成功的情况，应当不妨碍程序的继续运行
    }

    sqlite3_close(pdb);
    printf("table \"chaters\" has been created successfully.\n");
    return 0;
```

```c
}

int db_check(const char* username,const char* password,const char* dbname,sqlite3*
pdb){
    char* zerrmsg = NULL;
    char sql[100] = {0};
    char** pResult = NULL;
    int nRow = 0,nCol = 0;

    if(sqlite3_open(dbname,&pdb) != 0){
        printf("database can not be opened.\n");
        return -1;
    }

    sprintf(sql,"select * from chaters where username = '%s' and password = '%s';",
            username,password);
    int rc = sqlite3_get_table(pdb,sql,&pResult,&nRow,&nCol,&zerrmsg);
    if(rc != SQLITE_OK){
        sqlite3_close(pdb);
        printf("sql error: %s\n",zerrmsg);
        sqlite3_free(zerrmsg);
        return SQL_ERROR;
    }

    sqlite3_free_table(pResult);
    sqlite3_close(pdb);

    if(nRow == 0)
        return SQL_NONE;

    return SQL_FOUND;
}

int db_insert(const char* username,const char* password,const char* dbname,sqlite3*
pdb){
    char* zerrmsg = NULL;
    char sql[100] = {0};

    if(sqlite3_open(dbname,&pdb) != 0){
        printf("database can not be opened.\n");
        return -1;
    }

    sprintf(sql,"insert into chaters (username,password) values ('%s','%s');",
              username,password);

    int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
    if(rc != SQLITE_OK){
        sqlite3_close(pdb);
        printf("sql error: %s\n",zerrmsg);
        sqlite3_free(zerrmsg);
        return -1;
    }
    sqlite3_close(pdb);
    printf("%s inserted into table successfully.\n",username);
    return 0;
}

int db_delete(const char* username,const char* dbname,sqlite3* pdb){
    char* zerrmsg = NULL;
    char sql[100] = {0};

    if(sqlite3_open(dbname,&pdb) != 0){
        printf("database can not be opened.\n");
        return -1;
    }

    sprintf(sql,"delete * from chaters where username = '%s';",username);

    int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
    if(rc != SQLITE_OK){
        sqlite3_close(pdb);
        printf("sql error: %s\n",zerrmsg);
        sqlite3_free(zerrmsg);
        return -1;
    }
    sqlite3_close(pdb);
    printf("%s deleted from table successfully.\n",username);
    return 0;
}

static int callback(void* data,int argc,char** argv,char** azcolname){
    for(int i=0; i<argc; i++)
        printf("%s=%s\n",azcolname[i],argv[i]?argv[i]:NULL);
    printf("\n");

    return 0;
}
```

```

# ################### Makefile

default:
    gcc climain.c client.c -lpthread -o clnt
    gcc sermain.c server.c list.c sqlitedb.c -lpthread -lsqlite3 -o srvr
clean:
    rm *.o

# #################### mychat-v1.1 更新日志:

# 已解决问题:

# 1.彻底解决了消息收发过程中产生的乱码问题
# 问题原因:
#   read()函数将读取的消息写入缓冲区之后,并没有将有效数据后面紧跟的字节置为'\0',故对缓冲区直接进行字符串读取操作,可能会读取超过有效信息的部分,多出的部分就会变成乱码.
# 解决办法:
#   将有效信息之后紧跟的第一个字节置为'\0'字符,可以彻底解决read()函数造成的字符串接收乱码问题.

# 2.空白输入产生(null)广播消息的问题
# 问题原因:
#   没有对输入的文本进行有效性检查,导致无效信息(只包含"\n")被发送.其他客户端接收到无效信息,就会显示(null).
# 解决办法:
#   对输入内容进行有效性检查,缺乏有效信息的消息,将不予发送,并重新准备接收用户输入.

# 3.服务器不能正常退出,只能强制退出的问题
# 问题原因:
#   没有为服务器设置合理的退出办法,每次只能强制退出,导致退出之后端口仍然被占用,服务器不能正常重建.
# 解决办法:
#   为服务器设置单独的输入接收线程,当收到键盘输入:exit 的时候,服务器释放所有资源然后正常退出.

# 4.日志生成被覆盖的问题
# 问题原因:
#   没有对日志文件的名称进行差异化处理,导致每次开启程序,新的日志覆盖了旧的日志,并且一个客户端的日志覆盖了另一个客户端的日志.
# 解决办法:
#   a.对日志文件名插入客户名进行差异化处理,防止不同客户端生成的日志相互覆盖;
#   b.对当天生成的日志文件,文件名上加入当天的日期,既方便查找,也不会覆盖.
#   c.对聊天日志的格式进行了进一步的优化，采用 hh:mm:ss msg\n 格式，相比之前更加的简洁清晰。

# 5.用户重复进入聊天房间自己和自己聊天的问题
# 问题原因:
#   由于服务器规定进入聊天房间才能计入在线列表，所以删除了之前对重复登录进行检查的代码，而没有相应增加对重复进入聊天房间的检查代码。
# 解决办法:
#   在用户进入聊天房间的第一时间，服务器根据用户名进行在线状态检查，如果用户已经进入房间，则向客户端发送重复登入提示然后结束针对该用户的聊天服务，如果用户没有进入房间，则发送登入成功消息，然后将用户加入在线列表并提供转发服务。

# 6.客户端缺乏帮助功能，新用户不了解软件功能和使用方法的问题
# 解决办法:
#   增加了客户端帮助程序，在 command:栏输入 help，即可获得完整的命令列表和功能说明.

# 7.服务器测试提示语句过多，可能降低服务效率的问题
# 解决办法:
#   注释了服务器代码中大部分已经测试通过的功能模块的提示语句。

# 8.用户 login 登录成功之后，仍然可以 register，造成逻辑错误的问题
# 解决办法:
#   a.注册部分，增加登录状态检查，如果已经登录，则不允许注册，提示需要先登出。
#   b.添加登出功能。

# 9.文件传送乱码和中文文本显示不正常的问题
# 问题原因:
#   文件传送乱码或中文文本显示不正常,两者本质上是一个问题,都是文件转发过程中发生了意外修改,以及对缓冲区读取方式不当造成的.
# 解决办法:
#   a.服务器不再对客户消息作任何修改,停止对客户消息进行 strtok(msg,"\n")或添加\n 的操作.
#   b.对所有缓冲区读取得到的内容,根据有效信息长度,将最后一个有效字节后面紧跟的第一个字节设为'\0',
# 就将缓冲区字节数组转换成为标准的'\0'结尾字符串,然后再调用任何格式化字符串操作,都能获得预期的效果.

# 待解决问题:

# 1.在公网通信需要进行 ip 解析的问题
# 2.客户端界面和友好操作的问题
# ###################### mychat-v1.2 更新日志

# 已解决问题:

# 1.文件传送机制方方面面存在重大逻辑缺陷和流程错误的问题

#   问题原因:

#   a.数据包缺乏统一格式,导致内容解析流程复杂;
#   b.消息的来源/去向/单发/群发等必要信息获取不足;
#   c.部分条件下对数据包过度解析甚至修改,增加服务器负担,且容易造成信息失真;
#   d.在前版的文件收发机制下,文件收发命令可以单发或群发,但文件数据包的收发却始终是群发实现,属于严重逻辑错误;
#   e.因为 read()函数的返回次数不确定,所以通过数据包大小和文件大小来确定循环次数的方法,对接受端失效,引起接收循环次数不足,造成文件接收不完整和消息显示错误;
#   f.发送客户端和接收客户端之间,缺乏协调同步机制,导致接受端还没有进入准备接收状态,发送端就已经发送消息完毕,从而引起数据丢失和消息显示错误.

#   解决办法:

#   a.定义统一的数据收发格式,服务器不再对消息内容进行解析和改动,只进行群发和定向转发.
#   b.发送格式统一为@toname realmsg\n,接收格式统一为 fromname:@toname realmsg\n.服务器对 toname 进行判断,针对 toname 不存在/存在/="."等三种情况,分别进行错误返回/单向转发/群发.若发送方不指定@toname,则发送端自动添加@.至消息头,@.表示群发.
#   以上自定义消息协议,有效的解决了消息必要信息不足的问题,从而为更加精准/科学/有效的消息转发机制提供了可能.\
#   c.在统一消息格式下,服务器的工作方式更加简单,只需要根据来源客户端 cfd 找到对应的 fromname,然后添加 fromname:至要转发的消息头部,再根据 toname 的情况进行针对性的返回或转发.
#   d.在统一消息格式下,任何对话消息或文件流包,都会被添加 fromname:@toname 消息头,从而为所有数据流的正确定向提供了充分条件.指定接受人的文件流,将不会再被群发.
#   e.重新定义文件收发的循环退出机制,发送和接收统一设定为死循环.发送方统计每一次实际发出的具体字节数,累计达到文件大小,则退出循环;接受方统计每一次实际接受到的具体字节数,累计达到文件大小,则退出循环.这样直接杜绝了文件还没有收发完毕就退出循环的 bug.
#   f.针对文件收发不同步导致信息丢失和消息显示错误的问题,分两种情况处理.
#   第一种情况:
#   对定向单发的文件流,收发双方开始收发操作之前,接收方必须先发送接收状态认证到发送方,然后发送方根据状态认证分别进行处理.如果状态认证为[verify]: OK.\n 则表示获取文件 size 信息正常,可以进入收发循环;若状态认证为[verify]: NO.\n 表示获取文件 size 失败,取消本次文件传输.
#   收发循环开始后,每一次接收方都要发送接收状态认证,才可进行本次操作,若状态认证为[verify]: CC.\n 则表示可以继续进行下方操作;若认证状态为[verify]: SS.\n 则表示出现异常必须停止文件传输.
#   经过收发状态认证之后,接收方始终会先一步进入准备接收状态,就可以直接避免因为时间滞后而没有接收到数据包的问题.发送方必须获得接收认证,才可发送文件流包,否则进入阻塞等待状态.
#    以上收发方协调同步机制,清晰规范了文件收发对话流程,有效避免了文件收发不同步引起的数据丢失和显示错乱问题.
#    第二种情况:
#    对群发的文件流,发送方是唯一的,但是接收者是众多的,此时通过状态认证来实现的收发协调机制将失效.任何一个接收方的认证消息都可能解除发送方的阻塞状态,而发送方发送文件的时候,并不能保证所有的接收方都正确/及时地进入了接收状态.
#    所以对于群发的文件,必须创建新的解决方案.可行的方法是文件上传,通过:upload $filepath 语句,将本地文件上传至服务器,然后任何客户端都可以通过:download $filepath 命令来下载文件至本地.这样就完美解决了共享文件的异步收发问题.


# 2.@toname 之 toname 不在线,但文件传输依然被启动且进入阻塞状态的问题.

#    问题原因:
#    没有对 toname 进行在线状态检查

#    解决方案:
#    对 toname 进行在线状态检查之后,根据在线状态再决定是否开启文件传输和日志录入.



# 待解决问题:

# 1.共享文件上传至服务器,以及从服务器下载共享文件的问题.
# 2.在公网通信需要进行 ip 地址解析的问题
# 3.客户端界面和友好操作的问题
```