```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){
```

```
45          printf("Usage: clnt <ip>\n");
46          return -1;
47      }
48      int sfd = pconnect(argv[1]);
49  */
50      int sfd = pconnect("127.0.0.1");
51      if(sfd == -1){
52          printf("pconnect fails\n");
53          return -1;
54      }
55      printf("successfully connected.\n");
56
57      while(1){
58          switch(pcommand()){
59              case LOGIN:
60                  plogin(sfd);
61                  break;
62              case LOGOUT:
63                  plogout();
64                  break;
65              case REGISTER:
66                  pregister(sfd);
67                  break;
68              case CHECKON:
69                  pcheckon(sfd);
70                  break;
71              case TALK:
72                  ptalk(sfd);
73                  break;
74              case QUIT:
75                  pquit(sfd);
76                  break;
77              case HELP:
78                  phelp();
79                  break;
80              case UNKNOWN:
81                  punknown();
82                  break;
83              default:
84                  break;
85          }
86      }
87      return 0;
88  }
```

```
89
90  #include "client.h" ////////////////// client.c
91
92  char cmd[32] = {0};
93  int logstatus = 0;
94  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
95  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
96  pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
97  int ncond = 0;
98  FILE* pfile = NULL;
99  char myname[32] = {0};
100
101 int pcommand(void){
102
103     while(1){
104         printf("\ncommand:");
105         fflush(stdin);
106         fgets(cmd,20,stdin);//包含'\n'
107         if(strchr(cmd,' ')){
108             printf("space is not permitted in command.\n");
109             continue;
110         }
111         if(strlen(cmd) == 0){
112             printf("command can't be null.\n");
113             continue;
114         }
115         break;
116     }
117     if(!strcmp(cmd,"help\n"))
118         return HELP;
119     else if(!strcmp(cmd,"login\n"))
120         return LOGIN;
121     else if(!strcmp(cmd,"logout\n"))
122         return LOGOUT;
123     else if(!strcmp(cmd,"register\n"))
124         return REGISTER;
125     else if(!strcmp(cmd,"online\n"))
126         return CHECKON;
127     else if(!strcmp(cmd,"talk\n"))
128         return TALK;
129     else if(!strcmp(cmd,"quit\n"))
130         return QUIT;
131     else
132         return UNKNOWN;
```

```
133 }
134
135 void phelp(void){
136     printf("    login       set logstatus on;\n"
137            "    logout      set logstatus off\n"
138            "    register    register user ID;\n"
139            "    online      check online list;\n"
140            "    talk        enter talkroom;\n"
141            "    quit        quit this client.\n");
142 }
143
144 int psendcmd(int sfd){
145
146     strtok(cmd,"\n");
147     dprintf(sfd,"%s\n",cmd);
148 //  printf("command sent: %s\n",cmd);
149     return 0;
150 }
151
152 //注意:logstatus 是存储在客户端本地的登录状态判断条件,
153 //其状态是其他有关功能开启或终止的前提条件,
154 //但服务器仅将登入 chatroom 的用户计入 online list,
155 //以便更有针对性的统计处于聊天状态的实时有效用户。
156 //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
157 //plogin 成功后,logstatus 置为 1,否则 logstatus 等于初值 0
158 //plogout 将 logstatus 重新置为 0。
159
160 void plogout(void){
161     if(logstatus == 1)
162         logstatus = 0;
163
164     printf("logstatus off!\n");
165 }
166
167 int plogin(int sfd){
168
169     if(logstatus == 1){
170         printf("can not relogin, please retry.\n");
171         return -1;
172     }
173
174     if(psendcmd(sfd) == -1)
175         return -1;
176
```

```
177     char buf[100] = {0};
178     char username[32],password[32];
179     while(1){
180         printf("username:");
181         fgets(username,32,stdin);//包含'\n'
182         if(strchr(username,' ')){
183             printf("space is not permitted in username.\n");
184             continue;
185         }
186         if(strlen(username) > 24){
187             printf("username should be <= 24 characters\n");
188             continue;
189         }
190         break;
191     }
192 
193     while(1){
194         strcpy(password,getpass("password:"));//包含'\n'
195         if(strchr(password,' ')){
196             printf("space is not permitted in password.\n");
197             continue;
198         }
199         if(strlen(password) > 24){
200             printf("password should be <= 24 characters.\n");
201             continue;
202         }
203         break;
204     }
205     strtok(username,"\n");
206     strtok(password,"\n");
207     dprintf(sfd,"%s %s\n",username,password);
208     //全局变量 myname 赋值
209     strcpy(myname,username);
210 
211     int n = 0;
212     if((n = read(sfd,buf,100)) < 0 ){
213         printf("failed to read login reply from server.\n");
214         return -1;
215     }
216     buf[n] = '\0';
217 
218     if(strstr(buf,"successful")){
219         logstatus = 1;
220         printf("logstatus on!\n");
```

```
221     }else
222         printf("%s",buf);
223 
224     return 0;
225 }
226 
227 int pregister(int sfd){
228     if(logstatus == 1){
229         printf("please logout first!.\n");
230         return -1;
231     }
232 
233     if(psendcmd(sfd) == -1)
234         return -1;
235 
236     char buf[100] = {0};
237     char username[32],password[32];
238     while(1){
239         printf("username:");
240         fgets(username,32,stdin);//包含'\n'
241         if(strchr(username,' ')){
242             printf("space is not permitted in username\n");
243             continue;
244         }
245         if(strlen(username) > 24){
246             printf("username should be <= 24 characters\n");
247             continue;
248         }
249         if(!strcmp(username,".")){
250             printf("username should not be a '.' \n");
251             continue;
252         }
253         break;
254     }
255 
256     while(1){
257         strcpy(password,getpass("password:"));//包含'\n'
258         if(strchr(password,' ')){
259             printf("space is not permitted in password.\n");
260             continue;
261         }
262         if(strlen(password) > 24){
263             printf("password should be <= 24 characters.\n");
264             continue;
```

```
265         }
266         break;
267     }
268     char tmppass[32] = {0};
269     while(1){
270         strcpy(tmppass,getpass("confirm password:"));//包含'\n'
271         if(strchr(tmppass,' ')){
272             printf("space is not permitted in password.\n");
273             continue;
274         }
275         if(strlen(tmppass) > 24){
276             printf("password should be <= 24 characters.\n");
277             continue;
278         }
279         break;
280     }
281 
282     if(strcmp(password,tmppass)){
283         printf("password inputs differ,pleaes re_register.\n");
284         dprintf(sfd,"register failed\n");
285         return -1;
286     }
287 
288     strtok(username,"\n");
289     strtok(password,"\n");
290     dprintf(sfd,"%s %s\n",username,password);
291 
292     int n = 0;
293     if((n = read(sfd,buf,100)) < 0 ){
294         printf("failed to read register reply from server.\n");
295         return -1;
296     }
297     buf[n] = '\0';
298     printf("%s",buf);
299 
300     return 0;
301 }
302 
303 int pcheckon(int sfd){
304 
305     if(logstatus == 0){
306         printf("please login first!\n");
307         return -1;
308     }
```

```
309     if(psendcmd(sfd) == -1)
310         return -1;
311 
312     int cnt = 0;
313     char buf[16] = {0};
314     int n = 0;
315     if((n = read(sfd,buf,16)) <= 0)
316         printf("failed to get size of userlist.\n");
317     buf[n] = '\0';
318     sscanf(buf,"%d\n",&cnt);
319     printf("members online: %d\n",cnt);
320     if(cnt == 0) return 0;
321 
322     char* userlist = (char*)malloc(32*cnt+100);//彻底杜绝内存不够?32 不是已经够了吗
323     if(userlist == NULL){
324         printf("mem error:failed to malloc mem for userlist.\n");
325         return -1;
326     }
327     userlist[0] = '\0';
328     //printf("malloc done. starting reading userlist.\n");
329     //为什么服务器发送成功了,但是 read()函数经常不返回?
330     if((n = read(sfd,userlist,32*cnt+100)) <= 0){
331         printf("failed to get userlist from server.\n");
332         return -1;
333     }
334     userlist[n] = '\0';//如果内存不足的话,有可能设置字符串结尾\0 失败
335     //字符串\0 结尾设置不成功的话,就会无法正常输出
336     printf("%s\n",strtok(userlist,"\n"));//接收到的 userlist 自带\n
337     free(userlist);
338     userlist = NULL;
339 
340     return cnt;
341 }
342 
343 int ptalk(int sfd){
344 
345     if(logstatus == 0){
346         printf("please login first!\n");
347         return -1;
348     }
349     if(psendcmd(sfd) == -1)
350         return -1;
351 
352     char reply[128] = {0};
```

```
353     int r = 0;
354     if((r = read(sfd,reply,128)) < 0){
355         printf("failed to get reply from server!\n");
356         return -1;
357     }
358     reply[r] = '\0';
359     if(!strstr(reply,"successful")){
360         printf("%s\n",reply);
361         return -1;
362     }
363 
364     time_t t = time(NULL);
365     struct tm *today = localtime(&t);
366     char date[32] = {0};
367     sprintf(date,"%02d%02d%02d",today->tm_year+1900,today->tm_mon+1,today->tm_mday);
368 
369     char logname[256] = {0};
370     strcpy(logname,myname);
371     strcat(logname,"_chatlog_");
372     strcat(logname,date);
373     strcat(logname,".txt");
374 
375     pfile = fopen(logname,"a");
376     if(pfile == NULL)
377         printf("failed to open chatlog.\n");
378 
379     printf("\n");
380     pthread_t tid1,tid2;
381     if(pthread_create(&tid1,0,thread_send,(void*)&sfd) != 0){
382         dprintf(sfd,":exit\n");
383         printf("error: failed to create thread_send.\n");
384         return -1;
385     }
386     if(pthread_create(&tid2,0,thread_recv,(void*)&sfd) != 0){
387         dprintf(sfd,":exit\n");
388         printf("error : failed to create thread_recv.\n");
389         return -1;
390     }
391 
392     if(pthread_join(tid1,NULL) == 0 || pthread_join(tid2,NULL) == 0){
393         pthread_cancel(tid1);
394         pthread_cancel(tid2);
395     }
396 
```

```
397     fclose(pfile);
398     pfile = NULL;
399     return 0;
400 }
401 
402 void* thread_send(void* psfd){
403 
404     time_t t = 0;
405     struct tm *today = NULL;
406     int sfd = *(int*)psfd;
407     char msg[1000] = {0};
408     char filepath[100] = {0};
409     char toname[32] = {0};
410     char atme[32] = {0};
411     strcat(atme,"@");
412     strcat(atme,myname);
413 
414     while(1){
415         fgets(msg,1000,stdin);//包含\n\0
416 
417         if(strstr(msg,atme))
418             continue;
419         if(!strcmp(msg,"\n"))//空白消息,只包含\n 字符
420             continue;
421 
422         //规定群发@.之后，所有消息都带有@
423         if(msg[0] == '@'){//如果指定接收人，则修改 toname 为给定值;
424 
425             if(strstr(msg,":file")){
426                 sscanf(msg,"@%s",toname);
427                 if(strlen(toname) == 1 && toname[0] == '.'){
428                     printf("can not broadcast file by @.\n");
429                     continue;
430                 }
431 
432                 if(!strstr(msg,"$")){
433                     printf("$filepath should be designated.\n");
434                     continue;
435                 }
436                 sscanf(msg,"%*[^$]$%s",filepath);
437 
438                 dprintf(sfd,"%s",msg);//msg 包含@toname 和\n
439                 //首先判断 toname 是否存在,如果不存在,则返回
440                 pthread_mutex_lock(&mutex1);
```

```
441                 pthread_cond_wait(&cond,&mutex1);//经过通知,才能开始发送
442                 pthread_mutex_unlock(&mutex1);
443                 if(ncond != 1) continue;
444                 if(pfile_send(sfd,filepath,toname) == -1) continue;
445                 ncond = 0;
446             }else
447                 dprintf(sfd,"%s",msg);//msg 包含@toname 和\n
448         }else{//群发,补加@.
449             if(strstr(msg,":file")){//不允许进行文件群发
450                 printf("@toname should be designated.\n");
451                 continue;
452             }
453             dprintf(sfd,"@. %s",msg);//msg 包含\n
454         }
455 
456         t = time(NULL);
457         today = localtime(&t);
458         pthread_mutex_lock(&mutex);
459         fprintf(pfile,"%02d:%02d:%02d %s\n",today->tm_hour,today->tm_min,today->tm_s
460 ec,msg);
461         pthread_mutex_unlock(&mutex);
462         if(!strcmp(msg,":exit\n"))
463             return (void*)0;
464     }
465 }
466 
467 void* thread_recv(void* psfd){
468 
469     time_t t = 0;
470     struct tm *today = NULL;
471     int sfd = *(int*)psfd;
472     char msgbuf[1000] = {0};
473     char realmsg[1000] = {0};
474     char filepath[100] = {0};
475     char fromname[32] = {0};
476     char toname[32] = {0};
477     int lenfrom = 0;
478     int lento = 0;
479 
480     int n = 0;
481     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
482         if((n = read(sfd,msgbuf,1000)) <= 0){//若服务器退出，则退出
483             perror("read");
484             return (void*)-1;
```

```
485         }
486         msgbuf[n] = '\0';
487         //所有的消息格式都是 msgbuf = fromname:@toname realmsg
488         sscanf(msgbuf,"%[^:]",fromname);//:之前的所有字符
489         lenfrom = strlen(fromname);
490         sscanf(msgbuf,"%*[^@]@%s",toname);
491         lento = strlen(toname);
492         strcpy(realmsg,msgbuf+lenfrom+lento+3);
493     //printf("fromname=%s toname=%s realmsg=%s",fromname,toname,realmsg);
494 
495         //若对方确认接受文件,则设置 ncond 值
496         if(!strcmp(realmsg,"[verify]: OK.\n")){
497             ncond = 1;
498             pthread_cond_signal(&cond);
499         }
500         if(!strcmp(realmsg,"[verify]: NO.\n")){
501             ncond = 0;
502             pthread_cond_signal(&cond);
503         }
504         if(!strcmp(realmsg,"[verify]: CC.\n")){
505             ncond = 2;
506             pthread_cond_signal(&cond);
507         }
508         if(!strcmp(realmsg,"[verify]: SS.\n")){
509             ncond = -1;
510             pthread_cond_signal(&cond);
511         }
512         if(!strcmp(realmsg,"@toname not online!\n")){
513             ncond = -1;
514             pthread_cond_signal(&cond);
515         }
516 
517         if(!strstr(realmsg,"[verify]:")){//不显示[verify]:消息
518             //群发则不含@toname，realmsg 包含\n
519             if(strlen(toname) == 1 && toname[0] == '.')
520                 printf("%s:%s",fromname,realmsg);
521             else
522                 printf("%s:@%s %s",fromname,toname,realmsg);
523         }
524 
525         //此时 msg 不包含 fromname:@toname
526         if(strstr(realmsg,":file") && strstr(realmsg,"$")){
527             sscanf(realmsg,"%*[^$]$%s",filepath);
528             if(pfile_recv(sfd,filepath,fromname,toname) == -1){
```

```
529                 continue;//文件接收失败的话，接收请求就不写入日志
530             }
531         }
532 
533         t = time(NULL);
534         today = localtime(&t);
535         pthread_mutex_lock(&mutex);
536         fprintf(pfile,"%02d:%02d:%02d %s\n",today->tm_hour,today->tm_min,today->tm_s
537 ec,realmsg);
538         pthread_mutex_unlock(&mutex);
539     }
540 
541     return (void*)-1;
542 }
543 
544 int pfile_send(int sfd,char* filepath,char* toname){
545     printf("pfile_send: start sending..\n");
546 
547     //toname 最长 25 个字节
548     //解析文件名
549     char path[100] = {0};
550     char childpath[100] = {0};
551     char* name = NULL;
552     char cwd[100] = {0};
553     char tmpcwd[100] = {0};
554     char* curwd = NULL;
555     getcwd(cwd,100);
556     getcwd(tmpcwd,100);
557     int len = strlen(toname);
558 
559     //解析目标路径
560     if(strstr(filepath,"/")){
561         name = 1 + strrchr(filepath,'/');
562         strcpy(childpath,filepath);
563         //将 childpath 倒数第一个/ 设置为\0
564         strrchr(childpath,'/')[0] = '\0';
565     }else{
566         name = filepath;
567         strcpy(childpath,cwd);
568     }
569     printf("path=%s name=%s\n",childpath,name);
570 
571     //解析真实路径
572     //1 ~ home 目录起头
```

```
573     if(childpath[0] == '~'){
574         //1.1 有子目录
575         if(strlen(childpath) >1){
576             strcpy(path,getenv("HOME"));
577             strcat(path,strtok(childpath,"~"));
578             //strtok()一般情况下,将出现的字符全部设置为\0,
579             //然后返回剩下的字符串中不为\0 的首地址
580         }else
581             //1.2 没有子目录
582             strcpy(path,getenv("HOME"));
583     //2 / 根目录起头
584     }else if(childpath[0] == '/')
585         strcpy(path,childpath);
586     //3 .. 上层目录起头
587     else if(strlen(childpath) > 1 && childpath[0] == '.' && childpath[1] == '.'){
588         strcpy(path,cwd);//拷贝当前目录
589         //将倒数第一个/ 设置为\0 , 所得即是上层目录
590         strrchr(path,'/')[0] = '\0';
591         //3.1 有子目录
592         if(strlen(childpath) > 2)
593             //直接跳过.. 将后面的子目录连缀至上层路径 path
594             strcat(path,childpath+2);
595         //3.2 没有子目录
596             //什么都不干
597     //4 . 当前目录起头
598     }else if(childpath[0] == '.'){
599         //path 保存当前路径
600         strcpy(path,cwd);
601         //4.1 有子目录
602         if(strlen(childpath) > 1)
603             //跳过. 并连缀到当前目录
604             strcat(path,childpath+1);
605         //4.2 没有子目录
606             //啥都不干
607     //5 其他任意字符起头 通常表示当前目录下的子目录
608     }else{
609         strcpy(path,cwd);
610         strcat(path,"/");
611         strcat(path,childpath);
612     }
613     //路径解析完成
614     chdir(path);
615 //  printf("working directory changed as:%s\n",path);
616
```

```
617     //获取文件大小
618     int size = 0;
619     struct stat filestat = {0};
620     if(stat(name,&filestat) == -1){
621         dprintf(sfd,"@%s $staterr$\n",toname);
622         perror("stat error");
623         printf("\n");
624         return -1;
625     }
626     size = filestat.st_size;
627 
628     if(size == 0){
629         dprintf(sfd,"@%s $sizeerr$\n",toname);
630         printf("filesize=0,failed to send file.\n\n");
631         return -1;
632     }
633     //$file$在服务器转发过程中有特殊意义,
634     //表示以原字符串风格转发,不添加来源姓名
635     dprintf(sfd,"@%s filesize=%d\n",toname,size);
636 
637     //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
638     //所以需要通过 cond 条件变量,实现收发线程间的同步
639     pthread_mutex_lock(&mutex1);
640     pthread_cond_wait(&cond,&mutex1);
641     pthread_mutex_unlock(&mutex1);
642     if(ncond != 1){
643         printf("error: recver failed to recv file.\n");
644         return -1;
645     }
646 
647     FILE* psendfile = fopen(name,"r");
648     if(psendfile == NULL){
649         dprintf(sfd,"@%s $openerr$\n",toname);
650         perror("fopen error");
651         printf("\n");
652         return -1;
653     }
654 
655     int n = 0,w = 0;
656     int wsum = 0;
657     char filebuf[900] = {0};//不超过服务器接收范围
658     while(1){
659         pthread_mutex_lock(&mutex1);
660         pthread_cond_wait(&cond,&mutex1);//经过信号量通知,才能开始发送
```

```
661         pthread_mutex_unlock(&mutex1);
662         if(ncond != 2){
663             printf("error:file sending process failed.\n");
664             return -1;
665         }
666 
667         if((n = fread(filebuf,1,900,psendfile)) < 0){
668             ferror(psendfile);
669             return -1;
670         }
671         filebuf[n] = '\0';
672 
673         //如果不指定 toname,则 toname = ".";
674         w = dprintf(sfd,"@%s %s\n",toname,filebuf);//增加\n 以出尽缓存
675 
676         wsum += w-len-3;
677         printf("sent: %d bytes, %%%.2lf...\n",w-len-3,wsum*100.0/size);
678         ncond = 1;
679         if(wsum >= size)  break;
680     }
681 
682     ncond = 0;//发送完毕之后重置判断条件
683     fclose(psendfile);
684     psendfile = NULL;
685     chdir(cwd);
686     printf("file size=%d sent successful.\n\n",size);
687 
688     return 0;
689 }
690 
691 int pfile_recv(int sfd,char* filepath,char* fromname,char* toname){
692     char* name = NULL;
693     if(strstr(filepath,"/"))
694         name = 1 + strrchr(filepath,'/');
695     else
696         name = filepath;
697     printf("name=%s\n",name);
698 
699     //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
700 
701     //获取文件大小
702     int size = 0;
703     char sizebuf[64] = {0};
704     int n = 0;
```

```
705     if((n = read(sfd,sizebuf,32)) < 0){
706         perror("read error");
707         printf("\n");
708         return -1;
709     }
710     sizebuf[n] = '\0';
711     if(strstr(sizebuf,"$staterr$") || strstr(sizebuf,"$sizeerr$")){
712         printf("sender failed to fetch file size.\n\n");
713         return -1;
714     }
715     sscanf(sizebuf,"%*s filesize=%d\n",&size);
716 //  printf("size=%d\n",size);
717 
718     //根据文件大小,选择发送不同的认证消息
719 
720     if(size == 0){
721         dprintf(sfd,"@%s [verify]: NO.\n",fromname);
722         printf("filesize=0,failed to create file.\n\n");
723         return -1;
724     }
725 
726     dprintf(sfd,"@%s [verify]: OK.\n",fromname);
727 
728     FILE* precvfile = fopen(name,"w");
729     if(precvfile == NULL){
730         dprintf(sfd,"@%s [verify]: SS.\n",fromname);
731         perror("fopen error");
732         printf("\n");
733         return -1;
734     }
735 
736     int lenfrom = strlen(fromname);
737     int lento = strlen(toname);
738     int r = 0;
739     int w = 0;
740     int wsum = 0;
741     char filebuf[1000] = {0};
742     char realmsg[1000] = {0};
743     int lenreal = 0;
744     //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
745     while(1){
746         //通知发送方可以发送了
747         dprintf(sfd,"@%s [verify]: CC.\n",fromname);
748         r = read(sfd,filebuf,1000);//首先进入等待状态,阻塞接收
```

```
749         if(r < 0){//格式为 fromname:@toname realmsg\n
750             dprintf(sfd,"@%s [verify]: SS.\n",fromname);
751             perror("read error");
752             printf("\n");
753             ferror(precvfile);
754             return -1;
755         }
756         filebuf[r] = '\0';
757         //一共接收 r 个有效字符,
758         //格式为 fromname:@toname realmsg\n
759         strcpy(realmsg,filebuf+lenfrom+lento+3);
760         //绝对不能用 sscanf(),因为它遇空格或者换行就会停止
761 
762         if(strstr(realmsg,"$openerr$") || strstr(realmsg,"$readerr$")){
763             dprintf(sfd,"@%s [verify]: SS.\n",fromname);
764             printf("sender failed to send file content.\n\n");
765             return -1;
766         }
767         lenreal = strlen(realmsg);//包含\n
768         realmsg[lenreal-1] = '\0';// \n 替换为\0
769 
770         if((w = fwrite(realmsg,1,strlen(realmsg),precvfile)) < 0){
771             dprintf(sfd,"@%s [verify]: SS.\n",fromname);
772             ferror(precvfile);
773             return -1;
774         }
775 
776         wsum += w;
777         printf("recved: %d bytes, %%%.2lf...\n",w,wsum*100.0/size);
778         if(wsum >= size) break;
779     }
780 
781     fclose(precvfile);
782     precvfile = NULL;
783     printf("file size=%d recved successful.\n\n",size);
784 
785     return 0;
786 }
787 
788 int pquit(int sfd){
789 
790     psendcmd(sfd);
791     exit(0);
792 }
```

```
793
794 int punknown(void){
795
796     printf("command is not known,please reinput.\n");
797     return 0;
798 }
799
800 int pconnect(char* ip){
801
802     SA4 serv;
803     serv.sin_family = AF_INET;
804     serv.sin_port = htons(8080);
805     serv.sin_addr.s_addr = inet_addr(ip);
806
807     int sfd = socket(AF_INET,SOCK_STREAM,0);
808     if(sfd == -1){
809         perror("socket");
810         return -1;
811     }
812
813     int c = connect(sfd,(SA*)&serv,sizeof(serv));
814     if(c == -1){
815         perror("connect");
816         return -1;
817     }
818     return sfd;
819 }
820
821 ////////////////////// server.h
822 #ifndef _SERVER_H
823 #define _SERVER_H
824
825 #include <stdio.h>
826 #include <stdlib.h>
827 #include <unistd.h>
828 #include <string.h>
829 #include <sys/types.h>
830 #include <sys/socket.h>
831 #include <arpa/inet.h>
832 #include <pthread.h>
833 #include <sqlite3.h>
834 enum{LOGIN = 1,REGISTER,CHECKON,TALK,SENDFILE,QUIT};
835 enum{SQL_ERROR = -1,SQL_NONE,SQL_FOUND};
836
```

```
837 typedef struct sockaddr_in SA4;
838 typedef struct sockaddr SA;
839
840 typedef struct node{
841     char username[32];
842     int tcfd;
843     struct node* pprev;
844     struct node* pnext;
845 }node;
846
847 typedef struct list{
848     node* pcur;
849     node head;
850     node tail;
851 }list;
852
853 void* pexit(void*);
854 void* pnewthread(void* pcfd);
855 int pcommand(int cfd);
856 int plogin(int cfd,char** pmyname);
857 int pregister(int cfd);
858 int pcheckon(int cfd);
859 int ptalk_transfer(int cfd,char* myname);
860 void pgroupmsg(int mycfd,char* myname,char* msg);
861 int pquit(int cfd);
862 int plisten(int port,int backlog);
863
864 int list_init(list* plist);
865 int list_count(list* plist);
866 int list_show(list* plist,int cfd);
867 int list_getcfd(const char* username,list* plist);
868 int* list_getcfdarr(int** pcfdarr,int* pcnt,list* plist);
869 char* list_getname(int cfd,list* plist);
870 int list_append(const char* username,int cfd,list* plist);
871 int list_delete(int cfd,list* plist);
872 int list_destroy(list* plist);
873
874 int db_open(const char* dbname,sqlite3* pdb);
875 int db_check(const char* username,const char* password,const char* dbname,sqlite3*
876 pdb);
877 int db_insert(const char* username,const char* password,const char* dbname,sqlite3*
878 pdb);
879 int db_delete(const char* username,const char* dbname,sqlite3* pdb);
880 static int callback(void* data,int argc,char** argv,char** azcolname);
```

```
881
882 #endif//_SERVER_H
883
884 #include "server.h"////////////////// servermain.c
885
886 list users;
887 sqlite3* pdb;
888 const char* dbname = "chat.db";
889
890 int main(int argc,char** argv){
891     SA4 client;
892     socklen_t clilen = sizeof(client);
893
894     int sfd = plisten(8080,6);
895     if(sfd == -1){
896         printf("plisten failed.\n");
897         return -1;
898     }
899     printf("start listening ...\n");
900     //初始化在线用户链表
901     list_init(&users);
902     //创建数据库 并创建用户注册表
903     if(db_open(dbname,pdb) == -1)
904         return -1;
905
906     pthread_t tid0;
907     int ret = pthread_create(&tid0,0,pexit,NULL);
908     if(ret != 0){
909         printf("error %d: pthread_create failed.\n",ret);
910         return -1;
911     }
912
913     while(1){
914         char IP[32] = {0};
915         int cfd = accept(sfd,(SA*)&client,&clilen);
916         if(cfd == -1){
917             perror("accept");
918             return -1;
919         }
920
921         pthread_t tid;
922         int t = pthread_create(&tid,0,pnewthread,(void*)&cfd);
923         if(t != 0){
924             printf("error %d: pthread_create failed.\n",t);
```

```
925             return -1;
926         }
927         printf(/*"%s: */"client thread cfd=%d
928 created.\n",/*inet_ntop(AF_INET,&client.sin_addr,IP,32),*/cfd);
929     }
930 
931     return 0;
932 }
933 
934 
935 #include "server.h"////////////////// server.c
936 
937 extern list users;
938 extern sqlite3* pdb;
939 extern const char* dbname;
940 
941 void* pexit(void* null){
942     char cmd[32] = {0};
943     while(1){
944         fgets(cmd,32,stdin);//fgets()获取的字符串包含\n
945         if(!strcmp(cmd,":exit\n"))
946             exit(0);
947     }
948 }
949 
950 void* pnewthread(void* pcfd){
951 
952     char* myname = NULL;
953     int cfd = *(int*)pcfd;
954 
955     while(1){
956         switch(pcommand(cfd)){
957             case LOGIN:
958                 plogin(cfd,&myname);
959                 break;
960             case REGISTER:
961                 pregister(cfd);
962                 break;
963             case CHECKON:
964                 pcheckon(cfd);
965                 break;
966             case TALK:
967                 ptalk_transfer(cfd,myname);
968                 break;
```

```
969             case QUIT:
970                 pquit(cfd);
971                 break;
972             default:
973                 break;
974         }
975     }
976     return (void*)0;
977 }
978 
979 int pcommand(int cfd){
980 
981     char cmd[32] = {0};
982     int n = 0;
983     if((n = read(cfd,cmd,32)) < 0){
984         perror("read error");
985         return QUIT;//如果读不到 command,就会发出退出命令
986     }
987     cmd[n] = '\0';
988 
989     if(!strcmp(cmd,"login\n"))
990         return LOGIN;
991     else if(!strcmp(cmd,"register\n"))
992         return REGISTER;
993     else if(!strcmp(cmd,"online\n"))
994         return CHECKON;
995     else if(!strcmp(cmd,"talk\n"))
996         return TALK;
997     else if(!strcmp(cmd,"sendfile\n"))
998         return SENDFILE;
999     else if(!strcmp(cmd,"quit\n"))
1000            return QUIT;
1001
1002    return QUIT;
1003 }
1004
1005 int plogin(int cfd,char** pmyname){
1006
1007    char buf[100] = {0};
1008    char username[32] = {0},password[32] = {0};
1009
1010    int n = 0;
1011    if((n = read(cfd,buf,100)) < 0){
1012        printf("failed to read login message from client.\n");
```

```
1013            return -1;
1014        }
1015        buf[n] = '\0';
1016        sscanf(buf,"%s %s\n",username,password);
1017
1018        switch(db_check(username,password,dbname,pdb)){
1019            case SQL_NONE:
1020                dprintf(cfd,"username or password wrong!\n");
1021    //          printf("username or password wrong!\n");
1022                break;
1023            case SQL_FOUND:
1024                dprintf(cfd,"login successful!\n");
1025                *pmyname = (char*)malloc(32);
1026                strcpy(*pmyname,username);
1027    //          printf("%s cfd=%d login successful!\n",username,cfd);
1028                break;
1029            case SQL_ERROR:
1030                dprintf(cfd,"database currently unavailable,please retry later!\n");
1031                break;
1032            default:
1033                break;
1034        }
1035        return 0;
1036    }
1037
1038    int pregister(int cfd){
1039
1040        char buf[100] = {0};
1041        char username[32] = {0},password[32] = {0};
1042
1043        int n = 0;
1044        if((n = read(cfd,buf,100)) < 0){
1045            printf("failed to read register message from client.\n");
1046            return -1;
1047        }
1048        buf[n] = '\0';
1049        sscanf(buf,"%s %s\n",username,password);
1050        if(!strcmp(username,"register") && !strcmp(password,"failed")){
1051    //      printf("password inputs differ,client may retry.\n");
1052            return -1;
1053        }
1054
1055        switch(db_check(username,password,dbname,pdb)){
1056            case SQL_NONE...SQL_FOUND:
```

```
1057                if(db_insert(username,password,dbname,pdb) == 0){
1058                    if(db_check(username,password,dbname,pdb) == SQL_FOUND){
1059                        dprintf(cfd,"user registered successfully!\n");
1060    //                  printf("user registered successfully!\n");
1061                    }
1062                }else{
1063                    dprintf(cfd,"username already exists,please re_register!\n");
1064    //              printf("username already exists,please re_register!\n");
1065                }
1066                break;
1067            case SQL_ERROR:
1068                dprintf(cfd,"database currently unavailable,please retry later!\n");
1069                break;
1070            default:
1071                break;
1072        }
1073        return 0;
1074    }
1075
1076    int pcheckon(int cfd){
1077
1078        list_show(&users,cfd);
1079        return 0;
1080    }
1081
1082    int ptalk_transfer(int cfd,char* myname){
1083
1084        if(list_getcfd(myname,&users) > 0){
1085            dprintf(cfd,"relogin: user is online somewhere else!\n");
1086            printf("relogin: user is online somewhere else.\n");
1087            return -1;
1088        }else{
1089            dprintf(cfd,"enter talkroom successful.");
1090    //      printf("%d entered talkroom successful.\n",cfd);
1091        }
1092
1093        char msg[1000] = {0};
1094        int tcfd = 0;
1095        char toname[32] = {0};
1096        list_append(myname,cfd,&users);
1097
1098        int n = 0;
1099        int len = 0;
1100        while((n = read(cfd,msg,1000)) > 0){
```

```
1101            msg[n] = '\0';
1102            //msg 自带\n,尤其是文件内容,不能删掉
1103            if(!strcmp(msg,"@. :exit\n")){
1104                char exitmsg[100] = {0};
1105                sprintf(exitmsg,"@. [msg]:left talk.\n");
1106                pgroupmsg(cfd,myname,exitmsg);
1107                list_delete(cfd,&users);
1108                break;
1109            }
1110
1111            //所有消息格式都为@toname realmsg\n
1112            //组织成新的格式为 fromname:@toname realmsg\n
1113            sscanf(msg,"@%s ",toname);
1114            len = strlen(toname);
1115
1116            if(len == 1 && toname[0]=='.')//群发
1117                pgroupmsg(cfd,myname,msg);//包含@toname
1118            else{//单发
1119                if((tcfd = list_getcfd(toname,&users)) == -1){
1120                    dprintf(cfd,"server:@%s @toname not online!\n",myname);
1121                    continue;
1122                }
1123                if(strstr(msg,":file") && strstr(msg,"$"))//只对文件命令发送确认消息
1124                    dprintf(cfd,"server:@%s [verify]: OK.\n",myname);
1125                dprintf(tcfd,"%s:%s",myname,msg);//包含@toname
1126    //          printf("transfer realmsg len=%lu.\n",strlen(msg)-len-2);
1127            }
1128        }
1129        //printf("ptalk_transfer exited.\n");
1130        return 0;
1131    }
1132    void pgroupmsg(int mycfd,char* myname,char* msg){
1133
1134        int clients = 0;
1135        int* cfdarr = NULL;
1136        char toname[32] = {0};
1137        sscanf(msg,"@%s ",toname);
1138        int len = strlen(toname);
1139
1140        //该函数会调用 malloc 所以用完之后 一定要 free
1141        list_getcfdarr(&cfdarr,&clients,&users);
1142
1143        for(int i=0; i<clients; i++)
1144            if(cfdarr[i] != mycfd){
```

```
1145                dprintf(cfdarr[i],"%s:%s",myname,msg);//包含@toname
1146        //      printf("broadcast realmsg len=%lu.\n",strlen(msg)-len-2);
1147            }
1148
1149        //free 临时数组的内存
1150        free(cfdarr);
1151        cfdarr = NULL;
1152    }
1153
1154    int pquit(int cfd){
1155
1156        if(list_getname(cfd,&users))//如果 cfd 所对应的用户存在,则删除之
1157            list_delete(cfd,&users);//针对意外退出情况
1158        printf("client thread cfd=%d exited.\n",cfd);
1159        pthread_exit(NULL);
1160    }
1161
1162    int plisten(int port,int backlog){
1163
1164        SA4 serv;
1165        serv.sin_family = AF_INET;
1166        serv.sin_port = htons(port);
1167        serv.sin_addr.s_addr = htonl(INADDR_ANY);
1168
1169        int sfd = socket(AF_INET,SOCK_STREAM,0);
1170        if(sfd == -1){
1171            perror("socket");
1172            return -1;
1173        }
1174
1175        int b = bind(sfd,(SA*)&serv,sizeof(serv));
1176        if(b == -1){
1177            perror("bind");
1178            return -1;
1179        }
1180
1181        int l = listen(sfd,backlog);
1182        if(l == -1){
1183            perror("listen");
1184            return -1;
1185        }
1186
1187        return sfd;
1188    }
```

```
1189
1190
1191  #include "server.h" ////////////////////// list.c
1192
1193  int list_init(list* plist){
1194
1195      plist->pcur = NULL;
1196      plist->head.pprev = NULL;
1197      plist->tail.pnext = NULL;
1198      plist->head.pnext = &plist->tail;
1199      plist->tail.pprev = &plist->head;
1200
1201      printf("list_init successful.\n");
1202      return 0;
1203  }
1204
1205  int list_count(list* plist){
1206
1207      int cnt = 0;
1208      node* pnode = NULL;
1209      for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
1210          if(pnode != &plist->tail)
1211              cnt++;
1212
1213      return cnt;
1214  }
1215
1216  int list_show(list* plist,int cfd){
1217
1218      int cnt = list_count(plist);
1219      dprintf(cfd,"%d\n",cnt);
1220      if(cnt == 0) return 0;
1221
1222      char* userlist= (char*)malloc(32*cnt+100);//彻底杜绝内存不足? 32 不是已经够了吗?
1223      if(userlist == NULL){
1224          dprintf(cfd,"failed to get userlist.\n");
1225          printf("failed to get userlist.\n");
1226          return -1;
1227      }
1228
1229      //使用 strcat()之前，一定要 bzero.bzero 这个函数貌似经常出错
1230      userlist[0] = '\0';
1231
```

```
1232        node* pnode = NULL;
1233        int lensum = 0;
1234        //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
1235        for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext){
1236            strcat(userlist,pnode->username);
1237            strcat(userlist," ");
1238            lensum += strlen(pnode->username)+1;
1239        }
1240        userlist[lensum-1] = '\0';
1241        dprintf(cfd,"%s\n",userlist);//userlist 发出去之后包含\n
1242        //printf("userlist sent:%s\n",userlist);
1243        free(userlist);
1244        userlist = NULL;
1245        return 0;
1246    }
1247
1248    int list_getcfd(const char* username,list* plist){
1249
1250        node* pnode = NULL;
1251        for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
1252            if(!strcmp(username,pnode->username))
1253                return pnode->tcfd;
1254
1255        return -1;
1256    }
1257
1258    int* list_getcfdarr(int** pcfdarr,int* pcnt,list* plist){
1259        *pcnt = list_count(plist);
1260        *pcfdarr = (int*)malloc(sizeof(int) * (*pcnt));
1261        if(*pcfdarr == NULL){
1262            printf("failed to malloc mem to init cfdarr[clients].\n");
1263            return NULL;
1264        }
1265        int i = 0;
1266        node* pnode = NULL;
1267        for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
1268            (*pcfdarr)[i++] = pnode->tcfd;
1269
1270        return *pcfdarr;
1271    }
1272
1273    char* list_getname(int cfd,list* plist){
1274
1275        node* pnode = NULL;
```

```
1276        for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext)
1277            if(cfd == pnode->tcfd)
1278                return pnode->username;
1279
1280        printf("failed to get name from list where cfd=%d\n",cfd);
1281        return NULL;
1282    }
1283
1284    int list_append(const char* username,int cfd,list* plist){
1285
1286        node* pnode = (node*)malloc(sizeof(node));
1287        if(pnode == NULL){
1288            printf("\nfailed to append %s into list.\n",username);
1289            return -1;
1290        }
1291
1292        strcpy(pnode->username,username);
1293        pnode->tcfd = cfd;
1294        plist->tail.pprev->pnext = pnode;
1295        pnode->pprev = plist->tail.pprev;
1296        pnode->pnext = &plist->tail;
1297        plist->tail.pprev = pnode;
1298    //  printf("user cfd=%d appended to list successful.\n",cfd);
1299
1300        return 0;
1301    }
1302
1303    int list_delete(int cfd,list* plist){
1304
1305        node* pnode = NULL;
1306        for(pnode = plist->head.pnext; pnode != &plist->tail; pnode = pnode->pnext){
1307            if(cfd == pnode->tcfd){
1308                pnode->pprev->pnext = pnode->pnext;
1309                pnode->pnext->pprev = pnode->pprev;
1310                free(pnode);
1311                pnode = NULL;
1312    //          printf("\nuser cfd=%d deleted from list successful.\n",cfd);
1313                return 0;
1314            }
1315        }
1316        printf("user cfd=%d does not exist!\n",cfd);
1317        return -1;
1318    }
1319
```

```
1320  int list_destroy(list* plist){
1321
1322      plist->pcur = NULL;
1323      while(plist->head.pnext != &plist->tail){
1324          node* pfirst = &plist->head;
1325          node* pmid = pfirst->pnext;
1326          node* plast = pmid->pnext;
1327
1328          pfirst->pnext = plast;
1329          plast->pprev = pfirst;
1330          free(pmid);
1331          pmid = NULL;
1332      }
1333
1334      return 0;
1335  }
1336

1337  #include "server.h" ////////////////// sqlitedb.c
1338
1339  int db_open(const char* dbname,sqlite3* pdb){
1340      char* sql = NULL;
1341      char* zerrmsg = NULL;
1342
1343      if(sqlite3_open(dbname,&pdb) != 0){
1344          printf("database can not be opened.\n");
1345          return -1;
1346      }
1347
1348      //sql 主键只能有一个
1349      sql = "create table chaters(\n"
1350          "username varchar(36) primary key not null,\n"
1351          "password varchar(36) not null);";
1352
1353      int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
1354      if(rc != SQLITE_OK){
1355          sqlite3_close(pdb);
1356          printf("sql: %s\n",zerrmsg);
1357          sqlite3_free(zerrmsg);
1358          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
1359      }
1360
1361      sqlite3_close(pdb);
1362      printf("table \"chaters\" has been created successfully.\n");
1363      return 0;
```

```
1364  }
1365
1366  int db_check(const char* username,const char* password,const char* dbname,sqlite3*
1367  pdb){
1368      char* zerrmsg = NULL;
1369      char sql[100] = {0};
1370      char** pResult = NULL;
1371      int nRow = 0,nCol = 0;
1372
1373      if(sqlite3_open(dbname,&pdb) != 0){
1374          printf("database can not be opened.\n");
1375          return -1;
1376      }
1377
1378      sprintf(sql,"select * from chaters where username = '%s' and password = '%s';",
1379              username,password);
1380      int rc = sqlite3_get_table(pdb,sql,&pResult,&nRow,&nCol,&zerrmsg);
1381      if(rc != SQLITE_OK){
1382          sqlite3_close(pdb);
1383          printf("sql error: %s\n",zerrmsg);
1384          sqlite3_free(zerrmsg);
1385          return SQL_ERROR;
1386      }
1387
1388      sqlite3_free_table(pResult);
1389      sqlite3_close(pdb);
1390
1391      if(nRow == 0)
1392          return SQL_NONE;
1393
1394      return SQL_FOUND;
1395  }
1396
1397  int db_insert(const char* username,const char* password,const char* dbname,sqlite3*
1398  pdb){
1399      char* zerrmsg = NULL;
1400      char sql[100] = {0};
1401
1402      if(sqlite3_open(dbname,&pdb) != 0){
1403          printf("database can not be opened.\n");
1404          return -1;
1405      }
1406
1407      sprintf(sql,"insert into chaters (username,password) values ('%s','%s');",
```

```
1408            username,password);
1409
1410    int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
1411    if(rc != SQLITE_OK){
1412        sqlite3_close(pdb);
1413        printf("sql error: %s\n",zerrmsg);
1414        sqlite3_free(zerrmsg);
1415        return -1;
1416    }
1417    sqlite3_close(pdb);
1418    printf("%s inserted into table successfully.\n",username);
1419    return 0;
1420 }
1421
1422 int db_delete(const char* username,const char* dbname,sqlite3* pdb){
1423    char* zerrmsg = NULL;
1424    char sql[100] = {0};
1425
1426    if(sqlite3_open(dbname,&pdb) != 0){
1427        printf("database can not be opened.\n");
1428        return -1;
1429    }
1430
1431    sprintf(sql,"delete * from chaters where username = '%s';",username);
1432
1433    int rc = sqlite3_exec(pdb,sql,NULL,NULL,&zerrmsg);
1434    if(rc != SQLITE_OK){
1435        sqlite3_close(pdb);
1436        printf("sql error: %s\n",zerrmsg);
1437        sqlite3_free(zerrmsg);
1438        return -1;
1439    }
1440    sqlite3_close(pdb);
1441    printf("%s deleted from table successfully.\n",username);
1442    return 0;
1443 }
1444
1445 static int callback(void* data,int argc,char** argv,char** azcolname){
1446    for(int i=0; i<argc; i++)
1447        printf("%s=%s\n",azcolname[i],argv[i]?argv[i]:NULL);
1448    printf("\n");
1449
1450    return 0;
1451 }
```

```
1452
1453  # ################## Makefile
1454
1455  default:
1456      gcc climain.c client.c -lpthread -o clnt
1457      gcc sermain.c server.c list.c sqlitedb.c -lpthread -lsqlite3 -o srvr
1458  clean:
1459      rm *.o
1460
1461  # ################### mychat-v1.1 更新日志:
1462
1463  # 已解决问题:
1464
1465  # 1.彻底解决了消息收发过程中产生的乱码问题
1466  # 问题原因:
1467  #   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
1468  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.
1469  # 解决办法:
1470  #   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
1471  问题.
1472
1473  # 2.空白输入产生(null)广播消息的问题
1474  # 问题原因:
1475  #   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,
1476  就会显示(null).
1477  # 解决办法:
1478  #   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.
1479
1480  # 3.服务器不能正常退出,只能强制退出的问题
1481  # 问题原因:
1482  #   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
1483  常重建.
1484  # 解决办法:
1485  #   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
1486  出.
1487
1488  # 4.日志生成被覆盖的问题
1489  # 问题原因:
1490  #   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
1491  端的日志覆盖了另一个客户端的日志.
1492  # 解决办法:
1493  #   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;
1494  #   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
```

```
1538  # ###################### mychat-v1.2 更新日志
1539
1540  # 已解决问题:
1541
1542  # 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
1543
1544  #    问题原因:
1545
1546  #    a.数据包缺乏统一格式,导致内容解析流程复杂;
1547  #    b.消息的来源/去向/单发/群发等必要信息获取不足;
1548  #    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;
1549  #    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;
1551  #    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;
1553  #    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.
1555
1556  #    解决办法:
1557
1558  #    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.
1559  #    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.
1562  #    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.\
1564  #    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.
1566  #    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.
1568  #    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.
1571  #    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.
1572  #    第一种情况:
1573  #    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.
1576  #    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.
1579  #    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

```