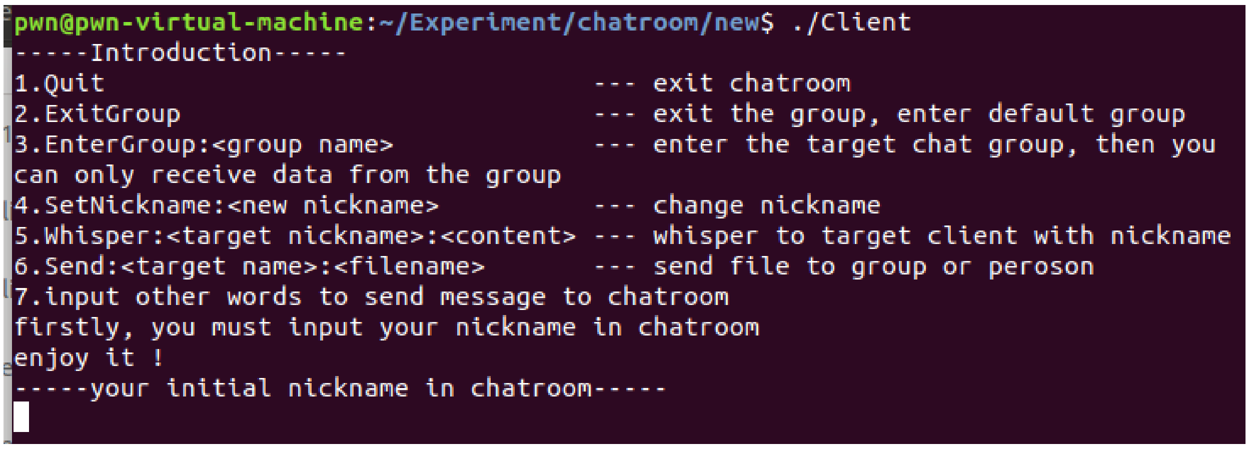
大纲

1. 本文基于聊天室的输入输出处理等。
2. 服务端部分主要内容为client_handler函数内容，即读取客户端信息、处理错误和退出请求、字符串分割、获取命令并进行相关处理、写回客户端等。
3. 客户端部分主要内容为main里的主循环中的读取用户输入并发给服务端

聊天室功能和特色

1. 支持多人在线聊天
2. 支持私聊
3. 支持群组聊天
4. 支持文件对点发送（不支持断点传输）
5. 更改昵称
6. 支持不同种信息以不同高对比度颜色呈现
7. 可查看当前聊天室（或群组）在线用户 Status

每个人在进入聊天室前需要初始化一个昵称，这个昵称之后也可以通过命令修改。



```
pwn@pwn-virtual-machine:~/Experiment/chatroom/new$ ./Client
-----Introduction-----
1.Quit                              --- exit chatroom
2.ExitGroup                         --- exit the group, enter default group
3.EnterGroup:<group name>           --- enter the target chat group, then you
can only receive data from the group
4.SetNickname:<new nickname>        --- change nickname
5.Whisper:<target nickname>:<content> --- whisper to target client with nickname
6.Send:<target name>:<filename>     --- send file to group or peroson
7.input other words to send message to chatroom
firstly, you must input your nickname in chatroom
enjoy it !
-----your initial nickname in chatroom-----
```

服务端

宏定义

```
#define PORT 14444
#define BUFFER_LENGTH 1024
#define MAX_CONN_LIMIT 512
#define CLIENT_SEND_PORT 14446
#define CLIENT_RECEIVE_PORT 14447
#define CLIENT_CTL_PORT 14445
```

用户的数据结构

用一个链表来存储所有客户端，客户端结点结构如下

```
typedef struct ClientNode{
    int socket_fd; //客户端聊天室的socket套接字
    int send_socket; //客户端发送文件的套接字
    int receive_socket; //客户端接收文件的套接字
    int ctl_socket;
    char nickname[50]; //昵称
    char group_name[50]; //所在群组名
    pthread_t thread_id;
    char Client_ip[512];
     struct ClientNode *next; //指向下一个用户
     struct ClientNode *root; //指向存储所有客户端的链表的头
```

```c
}Client;
```

客户端链表操作函数

```c
Client * init_client(){
 Client * p=(Client*)malloc(sizeof(Client));
 return p;
}
//p为原链表，elem表示新数据元素，add表示新元素要插入的位置
Client * insert_elem(Client * p, Client* elem, int add) {
 Client * temp = p;//创建临时结点temp
 //首先找到要插入位置的上一个结点
 for (int i = 1; i < add; i++) {
  temp = temp->next;
  if (temp == NULL) {
  printf("插入位置无效\n");
  return p;
  }
  }
  //向链表中插入结点
  elem->next = temp->next;
  temp->next = elem;
  return p;
 }
 //p为原链表，add为要删除元素的值
 Client * del_elem(Client * p, Client node) {
  Client * temp = p;
  //由于头节点的存在，因此while中的判断为t->next
  while (temp->next != NULL) {
  if (temp->next->socket_fd == node.socket_fd) break;
  temp=temp->next;
  }
  Client * del = temp->next;//单独设置一个指针指向被删除结点，以防丢
失
  temp->next = temp->next->next;//删除某个结点的方法就是更改前一个结
点的指针域
```

```c
 free(del);//手动释放该结点，防止内存泄漏
 return p;
}
int get_length(Client * p){
//新建一个指针t，初始化为头指针 p
 Client * t=p;
 int i=1;
 //由于头节点的存在，因此while中的判断为t->next
 while (t->next != NULL) {
 t=t->next;
 i++;
 }
 //程序执行至此处，表示查找失败
 return i;
}
```

自定义函数

1.改变字符串字体颜色

(将string的颜色格式化为color后存入字符串out中)

```c
enum Color {
      WHITE = 1, RED, GREEN, BLUE, YELLOW
};
void format_color_string(char* string, char* out, enum Color color){
    strcpy(out, "");
    switch (color)
    {
    case WHITE:
        strcat(out, "\033[37m");
        strcat(out, string);
        strcat(out, "\033[0m");
        break;
    case RED:
        strcat(out, "\033[31m");
```

```
            strcat(out, string);
            strcat(out, "\033[0m");
            break;
        case GREEN:
            strcat(out, "\033[32m");
            strcat(out, string);
            strcat(out, "\033[0m");
            break;
        case BLUE:
            strcat(out, "\033[34m");
            strcat(out, string);
            strcat(out, "\033[0m");
            break;
        case YELLOW:
            strcat(out, "\033[33m");
            strcat(out, string);
            strcat(out, "\033[0m");
            break;

        default:
            break;
        }
}
```

2.字符串分割

（将字符串src以separator为分隔符分割成若干子字符串分别存入字符串数组dest中，用num来记录子字符串的数量）

函数作用

根据自定义分隔符分割字符串

参数

src 源字符串的首地址(buf的地址)

separator 指定的分割符

dest 接收子字符串的字符串数组
num 分割后子字符串的个数
说明
用char*变量pNext作为子字符串的临时存储。用整型变量count临时记录子字符串数量。
char *strtok(char *str, const char *delim) 为C库函数中分解字符串的函数，其中 **str** 为一组字符串，**delim** 为分隔符。该函数返回被分解的第一个子字符串，如果没有可检索的字符串，则返回一个空指针。原字符串的改动是切分符原位置均更改为 **'\0'**，所以内容都还在。这个函数strtok(str,delim)隐含两个指针。
第一个指针pointer_a用来指向函数返回的字符串，这个字符串是被原字符串str被delim中的字符截断后的第一个字符串。
第二个指针pointer_b用来指向str中，匹配截断字串delim的位置。
NULL的作用只是为了使得每次调用时，都不是从str的头开始，而是从上次调用时查找所停止的位置pointer_b开始，如此循环下去，直到无法再找到匹配delim的位置。

```
void split(char *src,const char *separator,char **dest,int *num)
     char *pNext;
     int count = 0;
     //如果传入的地址为空或长度为0，直接终止
 if (src == NULL || strlen(src) == 0)
        return;

 //如未指定分割的字符串，直接终止
     if (separator == NULL || strlen(separator) == 0)
         return;
      pNext = (char *)strtok(src,separator);
  //必须使用(char *)进行强制类型转换
  //(虽然不写有的编译器中不会出现指针错误)
      while(pNext != NULL) {
            *dest++ = pNext;
            ++count;
           pNext = (char *)strtok(NULL,separator);  //必须使用(char *)进行强制类型转换
      }
      *num = count;
 }
```

3.查看聊天室用户状态

将要打印的信息存入out字符串中

```
void get_chatroom_status(Client* client, char* out){
 Client* t = client->root;
 sprintf(out, "-----ChatRoom Status-----\nclients count:%d\n-----clients-----\n", get_length(t) - 1);
 while (t->next != NULL) {
 t = t->next;
 strcat(out, "[");
 strcat(out, t->group_name);
 strcat(out, "]");
 strcat(out, t->nickname);
  strcat(out, "\n");
  }
```

函数作用

返回聊天室当前所有用户信息

参数

client：发起查询的用户结点

out：存储格式化后的查询到的聊天室的所有用户的信息

处理客户端请求的线程函数

对每一个连接的客户端新建一条线程来处理请求，该线程基于同一个静态函数创建.

```
static void* client_handler(void * p);
```

线程函数内通用局部变量定义

```
Client* client = ((Client *) p);
int i_reveive_bytes;
char data_recv[BUFFER_LENGTH];          //存 客户端发送的信息
char data_send[BUFFER_LENGTH];          //存 要发送给客户端的信息
char data_chatroom_send[BUFFER_LENGTH];//存 服务器最终发送给客户端
的格式化后的信息
```

线程函数中while循环处理客户端请求部分（主）

1.首先每次while循环前要先重制数据（data_recv,data_send,data_chatroom_send分别指收到的信息，用户发送的信息，聊天室系统发送的信息）

```
memset(data_recv, 0, BUFFER_LENGTH);
memset(data_send, 0, BUFFER_LENGTH);
memset(data_chatroom_send, 0, BUFFER_LENGTH);
```

2.先通过调用read函数从客户端的套接字读取用户输入，用i_reveive_bytes接收读取的字符长度，并将读取的数据存入data_recv中。读取后首先作报错捕捉。（即没有输入或读取输入出错）

```
        i_reveive_bytes = read(client->socket_fd, data_recv, BUFF
ER_LENGTH);  //不断读取客户端的输入请求
         if(i_reveive_bytes == 0)                     //没有读取到
客户端输入
         {
             printf("maybe the client has closed\n");
             break;
         }
         if(i_reveive_bytes == -1)                  //读取失败
         {
             fprintf(stderr, "read error!\n");
              break;
        }
```

3.确认读取客户端请求成功后，对客户端发来的字符串进行处理。字符指针数组revbuf用来存放分割后的子字符串。整型变量num用来存储分割后的子字符串个数。调用事先封装的split函数来以冒号为分隔符分割处理客户端发来的信息。

```
//如果读取到了输入
char* revbuf[4] = {0};      //存放分割后的子字符串
int num = 0;//分割后子字符串的个数
split(data_recv, ":", revbuf, &num);
```

4.这里再次给出聊天室的输入处理：
操作型输入：
发送文件给目标用户：Send:<TargetName>:<filename/path>
加入指定群组：EnteGroup:<GroupName>
和指定用户私聊：Whisper:<TargetName>:<Content>
更改用户昵称：SetNickname:<NewName>
退出当前群组：ExitGroup
退出聊天室：Quit
其他输入：
向当前群组发送信息

由于分割后的字符串会被分别存入revbuf中，如果找不到分隔符则revbuf[0]为原字符串，就是说，只要收到消息，字符串的个数num就会大于0.
我们可以首先根据分割得到的子字符串个数来初步判定用户输入的是指令级还是聊天信息。这样做的坏处显然是如果用户正常聊天内容中含有冒号则会被分割并判定为操作指令而不是聊天内容。改善的方式可以是将输入指令和聊天分割成两种输入模式，比如用户可以输入带指定特殊

符号的字符串来切换输入模式如【$SYSTEMMODE】【$CHATMODE】分别切换至指令操作模式和聊天模式。

言归正传，假如num>0，则说明用户有输入请求，通过调用strcmp函数来首先判定是不是"Quit"指令，如果是则不需要执行后续的代码块，会先通过服务器向跳出while循环执行退出聊天室后的善后操作：通过调用del_elm把当前用户从储存所有用户的链表中删除，关闭用户所在客户端与服务器的连接（套接字），并退出当前线程。

服务器向客户端发送的信息都会经过strcat和strcpy来格式化后通过封装后的format_color_string()函数将字体颜色变成对应的颜色并将变好颜色的字符串存入data_chatroom_send再通过对对应客户端socket调用write()发送给指定的客户端。格式化的过程后续将不再赘述。

```
while(1)
{
        //printf("waiting for request...\n");
        //Reset data.
        memset(data_recv, 0, BUFFER_LENGTH);
        memset(data_send, 0, BUFFER_LENGTH);
        memset(data_chatroom_send, 0, BUFFER_LENGTH);

        i_reveive_bytes = read(client->socket_fd, data_recv, BUFFER_LENGTH);  //不断读取客户端的输入请求
         if(i_reveive_bytes == 0)                    //没有读取到客户端输入
         {
             printf("maybe the client has closed\n");
             break;
         }
         if(i_reveive_bytes == -1)                   //读取失败
         {
             fprintf(stderr, "read error!\n");
```

```
            break;
        }

        //如果读取到了输入
        char* revbuf[4] = {0};     //存放分割后的子字符串

        //分割后子字符串的个数
        int num = 0;
        split(data_recv, ":", revbuf, &num);
        //调用函数对data_recv按冒号分割，分割后的子字符串存入revbuf，num为子字符串数量

        if (num > 0){   //说明读取到客户端输入
            if(strcmp(revbuf[0], "Quit") == 0)   //客户端输入Quit
            {
                printf("quit command!\n");
                strcpy(data_send, "[ChatRoom]");
                strcat(data_send, client->nickname);
                strcat(data_send, ":");
                strcat(data_send, "left the chatroom!");
                strcat(data_send, "\n");
                format_color_string(data_send, data_chatroom_send, BLUE);    //将data_send的字体转成蓝色存入data_chatroom_send
                Client* t = client->root;
                while (t != NULL) {  //广播这个用户退出的信息
                    write(t -> socket_fd, data_chatroom_send, strlen(data_chatroom_send));
                    t = t->next;
                }
                break;                           //Break the while loop.
            }
        }

        if(num == 3 && strcmp(revbuf[0],"Send") == 0){     //如果用户发起文件传输
```

```c
            Param *param = (Param*)malloc(sizeof(Param));
            memset(param->command,0,sizeof(param->command));
            for(int i=0;i<num;i++)strcpy(param->command[i],revbuf[i]);
            param->p = client;
            add_task(tpool,file_transport,(void*)param);
        }

        else if (num == 3){
            if(strcmp(revbuf[0], "Whisper") == 0){     //如果用户发起私聊
                char* target_nickname = revbuf[1];     //目标用户
                char* content = revbuf[2];             //私聊内容
                Client* t = client->root;
                while (t != NULL) {
                    if (strcmp(t->nickname, target_nickname) == 0){       //找到目标用户
                        strcpy(data_send, "[ChatRoom]Whisper from ");
                        strcat(data_send, client->nickname);
                        strcat(data_send, ":");
                        strcat(data_send, content);
                        strcat(data_send, "\n");
                        format_color_string(data_send, data_chatroom_send, YELLOW);    //格式化私聊的内容，转为蓝色字体
                        write(t -> socket_fd, data_chatroom_send, strlen(data_chatroom_send));//将格式化后的内容发送给目标客户端
                    }
                    t = t->next;
                }
            }
        }
        else if (num == 2){
            if (strcmp(revbuf[0], "EnterGroup") == 0){    //如果用户想加入群组，群组的名字在revbuf[1]
                strcpy(client->group_name, revbuf[1]);    //填补客户端的群组信息
                strcpy(data_send, "[ChatRoom]you have entered group:");
                strcat(data_send, client->group_name);
                strcat(data_send, "\n");
                format_color_string(data_send, data_chatroom_send, RED);  //格式化客户端加入群组信息
                write(client->socket_fd, data_chatroom_send, strlen(data_chatroom_send)); //用红色字体告知用户成功加入群组
            }
            if (strcmp(revbuf[0], "Init") == 0){         //用户初始化，输入Init:Nickname，以Nickname加入聊天室
                strcpy(client->nickname, revbuf[1]);
                Client* t = client->root;
                while (t != NULL) {                      //
                    strcpy(data_send, "[ChatRoom]");
                    strcat(data_send, client->nickname);
                    strcat(data_send, " entered chatroom!\n");
                    format_color_string(data_send, data_chatroom_send, BLUE);
                    write(t -> socket_fd, data_chatroom_send, strlen(data_chatroom_send));
                    t = t->next;
                }
            }
            if(strcmp(revbuf[0], "SetNickname") == 0)    //改昵称
            {
                strcpy(client->nickname, revbuf[1]);
                strcpy(data_send, "[ChatRoom]you have changed nickname:");
                 strcat(data_send, client->nickname);
                 strcat(data_send, "\n");
                 format_color_string(data_send, data_chatroom_send, RED);
                 write(client->socket_fd, data_chatroom_send, strlen(data_chatroom_send));   //红色字体通知用户更改成功
            }
        }
        else if (num == 1){
            if (strcmp(revbuf[0], "ExitGroup") == 0){        /
退出当前群组，则加入默认组
                strcpy(client->group_name, "DefaultGroup");
                strcpy(data_send, "[ChatRoom]you have exited g
oup, and entered default group\n");
                format_color_string(data_send, data_chatroom_se
nd, RED);
                write(client->socket_fd, data_chatroom_send, s
rlen(data_chatroom_send));
            }
            if (strcmp(revbuf[0], "Status") == 0){          //
返回当前聊天室在线用户列表

                get_chatroom_status(client, data_send);
                format_color_string(data_send, data_chatroom_se
nd, RED);
                write(client->socket_fd, data_chatroom_send, s
rlen(data_chatroom_send));
            }
            else{                              //无特殊命令的聊
天
                Client* t = client->root;
                //由于头节点的存在，因此while中的判断为t->next
                while (t != NULL) {
                    strcpy(data_send, "[ChatRoom]");
                    if (strcmp(client->group_name, "DefaultGrou
p") == 0) strcat(data_send, "");  //如果用户是默认群组，则不做操作
                    else {
                        strcat(data_send, "[");
                        strcat(data_send, client->group_name);
                        strcat(data_send, "]");
                    }
                    if (strcmp(client->group_name, t->group_na
e) == 0){   //找到同组的用户
```

```
                        if (t->socket_fd != client->socket_fd){
                            strcat(data_send, client->nickname);
                        }
                        else{
                            strcat(data_send, client->nickname);
                            strcat(data_send, "(you)");   //因为是广播，广播的时候发送信息的用户会有一个(you)标识
                        }
                        strcat(data_send, ":");
                        strcat(data_send, revbuf[0]);    //revbuf[0] 即要发送的信息内容
                        strcat(data_send, "\n");
                        format_color_string(data_send, data_chatroom_send, GREEN);
                        write(t -> socket_fd, data_chatroom_send, strlen(data_chatroom_send));  //聊天信息以绿色字体广播给同群组的成员客户端
                    }
                    t = t->next;
                }
            }
        }


        printf("---read from client---\nclient_id : %d\nnickname : %s\n", client->socket_fd, client->nickname);
        for(int i=0;i<num;i++)
        printf("content%d:%s\n",i+1,revbuf[i]);
    }
    //假如用户输入Quit退出聊天室，就会跳出while循环执行下面的代码
    del_elem(client->root, *client);     //将用户从用户列表中删除

    //Clear
    printf("terminating current client_connection...\n");
    close(client->socket_fd);           //将该用户的socket套接
字关闭.
    pthread_exit(NULL);   //terminate calling thread!
}
```

文件传输部分

```
typedef struct param{
 char command[4][50];
 Client *p;
}Param;

//pthread pool
typedef struct tpool_work_t{
 void*(*work)(void*);
 void *argu;
  struct tpool_work_t* next;
}tpool_work_t;
typedef struct tpool{
 int shutdown;
 int max_pthread;
 pthread_t *tid;
 tpool_work_t* work_head;
 pthread_cond_t queue_ready;
 pthread_mutex_t queue_lock;
}tpool_t;
tpool_t* tpool;
void* work_routine(void *argu)
{
 tpool_t *tpool = (tpool_t*)argu;
 tpool_work_t *job = NULL;
 while(1)
 {
 pthread_mutex_lock(&(tpool->queue_lock));
 while(!(tpool->work_head) && !(tpool->shutdown)){
```

```c
 pthread_cond_wait(&(tpool->queue_ready),&(tpool->queue_lock));
 }
 if(tpool->shutdown){
 pthread_mutex_unlock(&(tpool->queue_lock));
 pthread_exit(NULL);
 }
 job = tpool->work_head;
 tpool->work_head = (tpool_work_t*)tpool->work_head->next;
 pthread_mutex_unlock(&(tpool->queue_lock));

 job->work(job->argu);
 free(job);
 }
 pthread_exit(NULL);
}
tpool_t* create_pool(int max_pthread)
{
 tpool_t *tpool = (tpool_t*)malloc(sizeof(tpool_t));
 tpool->shutdown = 0;
 tpool->max_pthread = max_pthread;
 tpool->tid = (pthread_t*)malloc(sizeof(pthread_t)*max_pthread);
 tpool->work_head = NULL;
 pthread_mutex_init(&(tpool->queue_lock),NULL);
 pthread_cond_init(&(tpool->queue_ready),NULL);
 for(int i=0;i<max_pthread;i++)
 pthread_create(&(tpool->tid[i]),NULL,work_routine,(void*)tpool);
 return tpool;
}
int add_task(tpool_t *tpool,void*(routine)(void*),void* argu)
{
 if(routine == NULL || tpool == NULL)return -1;
 tpool_work_t* tpool_work = (tpool_work_t*)malloc(sizeof(tpool_work_t));
 tpool_work->work = routine;
 tpool_work->argu = argu;
 tpool_work->next = NULL;
 pthread_mutex_lock(&(tpool->queue_lock));
 if(tpool->work_head == NULL)tpool->work_head = tpool_work;
 else{
 tpool_work_t *p = tpool->work_head;
 while(p->next != NULL)p = p->next;
 p->next = tpool_work;
 }
 pthread_cond_signal(&(tpool->queue_ready));
 pthread_mutex_unlock(&(tpool->queue_lock));
 return 1;
}
void destory_tpool(tpool_t *tpool)
{
 tpool_work_t *tmp;
 if(tpool->shutdown == 1)return ;
 tpool->shutdown = 1;
 pthread_mutex_lock(&(tpool->queue_lock));
 pthread_cond_broadcast(&(tpool->queue_ready));
 pthread_mutex_unlock(&(tpool->queue_lock));
 for(int i=0;i<tpool->max_pthread;i++)
 pthread_join(tpool->tid[i],NULL);
 free(tpool-> tid);
 while(tpool->work_head)
 {
 tmp = tpool->work_head;
 tpool->work_head = tpool->work_head->next;
 free(tmp);
 }
 pthread_mutex_destroy(&(tpool->queue_lock));
 pthread_cond_destroy(&(tpool->queue_ready));
 free(tpool);
}
void* file_transport(void *argu)
```

```c
{
 Param* param = (Param*)argu;
 struct sockaddr_in sockaddr;
  socklen_t socklen;
  int send_feedback = accept(param->p->send_socket,(struct sockaddr *)&sockaddr,&socklen);
  //printf("after accpet");
  if(send_feedback < 0){
  perror("accpet error");
  free(param);
  return NULL;
  }
  int receive_fd[MAX_CONN_LIMIT],num=0,flag;
  int afd;
  if(strcmp(param->command[1],"Group") == 0)
  {
  Client *p = param->p->root->next;
  while(p != NULL)
  {
  if(strcmp(param->p->group_name,p->group_name) == 0 && strcmp(param->p->nickname,p->nickname) != 0)
  {
  send(p->ctl_socket,"Receive",8,0);
  afd = accept(p->receive_socket,(struct sockaddr*)&sockaddr,&socklen);
  if(afd > 0)receive_fd[num++] = afd;
  }
  p = p->next;
  }
  }
  else{
  Client *p = param->p->root->next;
  while(p!= NULL && strcmp(p->nickname,param->command[1]) != 0)
  p = p->next;
  if(p == NULL){
  printf("No such person.\n");
  flag = 3;
  send(send_feedback,&flag,sizeof(flag),0);
  free(param);
  return NULL;
  }
  else{
  send(p->ctl_socket,"Receive",8,0);
  afd = accept(p->receive_socket,(struct sockaddr*)&sockaddr,&so
  cklen);
  if(afd >0)receive_fd[num++] = afd;
  }
  }
  // printf("%d\n",num);
  if(!num){
  flag = 0;
  send(send_feedback,&flag,sizeof(flag),0);
  shutdown(send_feedback,SHUT_RDWR);
  free(param);
  printf("Send unsuccessfully!\n");
  return NULL;
  }
  flag = 1;
  char buff[BUFFER_LENGTH],command[BUFFER_LENGTH];
  memset(buff,0,sizeof(buff));
  memset(command,0,sizeof(command));
  // printf("before send\n");
  ssize_t bytes = send(send_feedback,&flag,sizeof(flag),0);
  if(bytes < 0){
  perror("send error");
  }
  //printf("%d\n",bytes);
  struct stat stat_buf;
  int cur_bytes = 0,ret;
  bytes = recv(send_feedback,&stat_buf,sizeof(stat_buf),0);
  sprintf(command,"%s:%s",param->p->nickname,param->command[2])
```

```
// printf("%s\n",command);
for(int i=0;i<num;i++)
send(receive_fd[i],&command,sizeof(command),0);
for(int i=0;i<num;i++)
send(receive_fd[i],&stat_buf,sizeof(stat_buf),0);
while(1)
{
bytes = recv(send_feedback,buff,sizeof(buff),0);
cur_bytes += bytes;
if(bytes < 0){
perror("recv error");
flag = 3;
for(int i=0;i<num;i++)shutdown(receive_fd[i],SHUT_WR);
send(send_feedback,&flag,sizeof(flag),0);
free(param);
return NULL;
}
else if(!bytes){
if(cur_bytes == stat_buf.st_size){
printf("Transport Finish!\n");
flag = 2;
send(send_feedback,&flag,sizeof(flag),0);
for(int i=0;i<num;i++)shutdown(receive_fd[i],SHUT_WR);
free(param);
return NULL;
}
else{
printf("Transport Interupted!\n");
}
flag = 3;
send(param->p->socket_fd,&flag,sizeof(flag),0);
for(int i=0;i<num;i++)shutdown(receive_fd[i],SHUT_WR);
free(param);
return NULL;
}
```

```c
else{
for(int i=0;i<num;i++)
{
ret = send(receive_fd[i],buff,bytes,0);
if(ret < 0){
printf("Somebody fail to receive\n");
flag = 3;
send(send_feedback,&flag,sizeof(flag),0);
for(int i=0;i<num;i++)shutdown(receive_fd[i],SHUT_WR);
free(param);
return NULL;
}
}
}
}
}
```

主函数部分

部分变量定义

```c
int total_connection_count = 0; //当前连接用户总数
int sockfd_server,
 result,
 sockfd_send,
 sockfd_receive,
 sockfd_ctl;

 int accept_feedback,
 send_feedback,
  receive_feedback,
  ctl_feedback;

  char buf[BUFSIZ],
  client_IP[1024];
```

socket配置

其中聊天室基本功能只用到sockfd_server和sockfd_ctl，而文件传输需要用到sockfd_send和sockfd_receive;

以聊天室基本功能的socket配置为例，基本流程：

1.配置sockaddr_in类型的包含地址族和端口的服务器信息server_address和客户端信息

2.调用socket()建立ipv4协议的套接字sockfd_server

2.调用bind()将配置好的server_address绑定在套接字sockfd_server上

3.调用listen()监听sockfd_server的被连接

4.调用socket()建立ipv4协议的套接字sockfd_clt，用于指代连接到服务器的客户端流

5.调用accept()返回

```c
struct sockaddr_in server_address, client_address;
struct sockaddr_in send_address, receive_address, ctl_address;
socklen_t clt_addr_len;
ctl_address.sin_family = AF_INET;
ctl_address.sin_port = htons(CLIENT_CTL_PORT);
ctl_address.sin_addr.s_addr = htonl(INADDR_ANY);

server_address.sin_family = AF_INET;
server_address.sin_port = htons(PORT);
server_address.sin_addr.s_addr = htonl(INADDR_ANY);

sockfd_server = socket(AF_INET,SOCK_STREAM,0);
sockfd_send = socket(AF_INET,SOCK_STREAM,0);
sockfd_receive = socket(AF_INET,SOCK_STREAM,0);
sockfd_ctl = socket(AF_INET,SOCK_STREAM,0);

 if(sockfd_server < 0) err("server_socket error\n");
if(sockfd_send < 0) err("send_socket error\n");;
if(sockfd_receive < 0) err("receive_socket error\n");
if(sockfd_ctl < 0) err("ctl_socket error\n");
result = bind(sockfd_server,(struct sockaddr*)&server_address,sizeof(server_address));
if(result < 0) err("server bind error\n");
result = bind(sockfd_ctl,(struct sockaddr*)&ctl_address,sizeof(ctl_address));
send_address.sin_family = AF_INET;
send_address.sin_port = htons(CLIENT_SEND_PORT);
send_address.sin_addr.s_addr = htonl(INADDR_ANY);
result = bind(sockfd_send,(struct sockaddr*)&send_address,sizeof(send_address));
if(result < 0) err("send bind error\n");

receive_address.sin_family = AF_INET;
receive_address.sin_port = htons(CLIENT_RECEIVE_PORT);
receive_address.sin_addr.s_addr = htonl(INADDR_ANY);
result = bind(sockfd_receive,(struct sockaddr*)&receive_address,sizeof(receive_address));
if(result < 0) err("receive bind error\n");

result = listen(sockfd_server,128);
result = listen(sockfd_send,128);
result = listen(sockfd_receive,128);
result = listen(sockfd_ctl,128);
if(result < 0) err("listen error\n");
```

```

Client *clients = init_client();
clients->next = NULL;


memset(client_IP,0,sizeof(client_IP));



```

```c
tpool = create_pool(5); //创建线程池

while (1) {
pthread_t thread_id;
clt_addr_len = sizeof(client_address);
accept_feedback = accept(sockfd_server, (struct sockaddr*) &client_address, &clt_addr_len);
if(accept_feedback < 0) err("accept error\n");
// send_feedback = accept(sockfd_send,NULL,NULL);
// if(send_feedback < 0) err("send accept error\n");
ctl_feedback = accept(sockfd_ctl,(struct sockaddr*)&receive_address,&clt_addr_len);
if(receive_feedback < 0) err("receive accept error\n");
printf("client IP: %s ,port:%d\n",
inet_ntop(AF_INET, &client_address.sin_addr.s_addr, client_IP, sizeof(client_IP)),
ntohs(client_address.sin_port)
);

pthread_attr_t attr;
pthread_attr_init( &attr );
pthread_attr_setdetachstate(&attr,1);

Client* new_client = (Client*)malloc(sizeof(Client));
new_client->socket_fd = accept_feedback; //即加入聊天室对应的socket
new_client->send_socket = sockfd_send;
new_client->receive_socket = sockfd_receive;
new_client->root = clients; //每个用户结点的root都是clients的头结点
new_client->ctl_socket = ctl_feedback;
strcpy(new_client->Client_ip,client_IP);
strcpy(new_client->group_name, "DefaultGroup");
strcpy(new_client->nickname, "DefaultNickName");


if (pthread_create(&thread_id, &attr, client_handler, (void *)
(new_client)) == -1){
err("thread create error\n");
break;
}

insert_elem(clients, new_client, get_length(clients));
new_client->thread_id = thread_id;
total_connection_count += 1;

printf("total_connection_count:%d\n", total_connection_count);

}

result = shutdown(sockfd_server, SHUT_WR); //关闭聊天室
assert (result != -1);

printf("server shut down!\n");
destory_tpool(tpool); //销毁线程池
return 0;
}
```

客户端

自定义函数

清除缓冲区（功能类似fflush(stdout))

```
void safe_flush(FILE *fp)
{
    int ch;
    while ((ch = fgetc(fp)) != EOF && ch != '\n');
}
```

为什么要清除缓冲区？因为在调用scanf()时是以回车结束的，此时系统会将回车读入输入缓冲区。在下次调用scanf()函数时，输入缓冲区中的回车就会先被读取。则在用户还没输入时scanf函数就已经先被缓冲区中的回车输入结束了。
该函数实现方法为遍历一遍缓冲区。

读取输入

```c
void read_input(char *buff)
{
    memset(buff,0,strlen(buff));
    scanf("%[^\n]", buff);
}
```

scanf中的“%[^\n]”是scanf的正则用法。因为scanf()函数不能接收空格字符，一接收空格字符就会结束读入。而使用“%[^\n]”格式化就可以使其一直读入直至读到换行符‘\n’。

文件名分割

```c
char* split_filename(char* filepath)
{
    if(filepath == NULL)return NULL;
    char *name_start = strchr(filepath,'/');
    if(name_start == NULL)return filepath;
    else name_start += 1;
    char *tmp = name_start;
    while((tmp = strchr(name_start,'/')) != NULL)
    {
         name_start = tmp+1;
     }
     return name_start;
 }
```

char *strchr(const char *str, int c) 在参数 **str** 所指向的字符串中搜索并返回第一次在字符串 str 中出现字符 c 的位置，如果未找到该字符则返回 NULL。

分割命令

```c
int split_command(char *command,char **rev)
{
    memset(rev,0,sizeof(rev));
    char *spliter = ":";
    int num = 0;
    char *tmp = NULL;
    tmp = strtok(command,spliter);
    while(tmp != NULL)
    {
         *(rev+num) = tmp;
         num++;
         tmp = strtok(NULL,spliter);
     }
     return num;
}
```

按冒号为分隔符分割用户输入的字符串command，将子字符串悉数存入rev中。

发送文件

准备

```c

typedef struct Param{
    char *filename;
    char filepath[100];
    char receiver[50];
}Param;
struct DataReceiverPara {
    int socket_fd;
};
```

通过不同于聊天室端口的发送文件端口连接至服务器进行发送文件

```c
void send_file(Param *param)
{
    printf("Beginning send_file\n");
    int ret=0,afd=0,fd=0,flag=0;
    struct stat fd_stat;
    //connect to server
    struct sockaddr_in sockaddr;
    sockaddr.sin_family = AF_INET;
    sockaddr.sin_port = htons(CLIENT_SEND_PORT);
     inet_pton(AF_INET,SERVER_IP,&sockaddr.sin_addr.s_addr);
     afd = socket(AF_INET,SOCK_STREAM,0);
     if(afd <0){
         perror("From sending process->Connection Error");
         return ;
     }
     ret = connect(afd,
(const struct sockaddr*)&sockaddr,sizeof(sockaddr));
     if(ret < 0){
         perror("From sending process->Connection Error");
         free(param);
         return ;
     }

  ret = stat(param->filename,&fd_stat);
     if(ret < 0){
         perror("get file attribute error");
         return ;
     }
     fd = open(param->filepath,O_RDONLY);
     if(fd < 0){
         perror("From sending process->open file error:");
         close(afd);
```

```c
        return ;
    }
    //receive reply from server
    ret = recv(afd,&flag,sizeof(flag),0);
    if(ret < 0){
        perror("From sending process->Receive Reply Fail");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    else if(!ret){
        out_of_connect("From sending process->");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    //accept and sendfile
    if(!flag){
        printf("From sending process->Fail to Connect Receiver or may be you are the only one in your group!\n");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    if(flag == 3){
        printf("From sending process->No such person!\n");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    ret = send(afd,&fd_stat,sizeof(fd_stat),0);
    ssize_t sz = sendfile(afd,fd,NULL,fd_stat.st_size);//zero copy
    shutdown(afd,SHUT_WR);
    if(sz < 0)perror("From sending process->send_file Error");
    //receive reply from server in order to know whether send successfully
    ret = recv(afd,&flag,sizeof(flag),0);
   // printf("After recv\n");
    if(ret < 0){
        perror("From sending process->Receive Reply Fail");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    else if(!ret){
        out_of_connect("From sending process->");
        close(afd);
        close(fd);
        free(param);
        return ;
    }
    if(flag == 2){
        printf("From sending process->File has been sent to %s successfully!\n",param->receiver);
    }
    else{
        printf("From sending process->send_file unsuccessfully!\n");
    }
    close(afd);
    close(fd);
    free(param);
    return ;
}
```

接收文件

```c
void data_path(int afd)
{
    printf("Receiving\n");
    char buff[BUFF_SIZE],command[BUFF_SIZE],*rev[2];
    memset(command,0,sizeof(command));
    memset(buff,0,sizeof(buff));
    int recv_bytes,cur_bytes=0;
    struct stat stat_buf;
    recv_bytes = recv(afd,command,sizeof(command),0);
     recv_bytes = recv(afd,&stat_buf,sizeof(stat_buf),0);
     if(recv_bytes < 0){
         perror("From receive process->Recieve file attribute error");
         close(afd);
         return ;
     }
     else if(!recv_bytes){
         out_of_connect("From receive process->");
         close(afd);
         return ;
     }
     /*
         command format sender:filename
     */
     split_command(command,rev);
     int fd = open(rev[1],O_WRONLY|O_CREAT,0644);
     printf("From receive process->Receiveing file from %s,total %d bytes\n"
     ,rev[0],(int)stat_buf.st_size);
     while(1)
     {
         recv_bytes = recv(afd,buff,sizeof(buff),0);
         cur_bytes += recv_bytes;
         if(recv_bytes < 0){
            perror("\nFrom receive process->Recieve filie error");
            close(afd);
            close(fd);
            return ;
        }
        else if(!recv_bytes){
            if(cur_bytes == stat_buf.st_size){
                close(afd);
                close(fd);
                printf("\nFrom receive process->Receive successfully\n");
                return ;
            }
        }
        else{
            write(fd,buff,recv_bytes);
            process_stat(cur_bytes,stat_buf.st_size);
        }
    }
    close(afd);
    return ;
}
```

全局定义

```c
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<pthread.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
 #include<sys/stat.h>
```

```c
#include<fcntl.h>
#include<sys/sendfile.h>
#include<net/if.h>
#include<sys/ioctl.h>
#include<sys/wait.h>
#include<signal.h>
#include<sys/epoll.h>

#define BUFF_SIZE 1024
#define IF_NAME "enp0s3"
#define CLIENT_SEND_PORT 14446 //发送文件的端口
#define CLIENT_RECEIVE_PORT 14447 //接收文件的端口
#define SERVER_IP "59.110.53.159" //聊天室的服务器
#define CLIENT_CTL_PORT 14445
#define CLIENT_CHAT_PORT 14444 //聊天室用的端口

socklen_t socklen = sizeof(struct sockaddr_in);
```

主函数部分

socket配置

配置ip地址（端口，协议族），创建聊天室和文件传输的socket套接字以及连接到指定服务器的配置。

```c
struct sockaddr_in sockaddr,ctl_addr;
sockaddr.sin_family = ctl_addr.sin_family = AF_INET;
sockaddr.sin_port = htons(CLIENT_CHAT_PORT);
ctl_addr.sin_port = htons(CLIENT_CTL_PORT);
inet_pton(AF_INET,SERVER_IP,&sockaddr.sin_addr.s_addr);
inet_pton(AF_INET,SERVER_IP,&ctl_addr.sin_addr.s_addr);
int afd = socket(AF_INET,SOCK_STREAM,0);
int ctl_fd = socket(AF_INET,SOCK_STREAM,0);
if(afd < 0)perror("socket error");

 int ret = connect(afd,
(const struct sockaddr*)&sockaddr,sizeof(sockaddr));
```

```c
ret = connect(ctl_fd,(const struct sockaddr*)&ctl_addr,sizeof(ctl_addr));
if(ret < 0){
 perror("connect error");
     return -1;
}
```

线程配置

创建文件传输线程（该线程会在文件传输fd收到消息是创建文件接收线程，该线程内会连接服务端文件接收端口），

```c
pthread_t tid;
pthread_attr_t pth_attr;
pthread_attr_init(&pth_attr);
pthread_attr_setdetachstate(&pth_attr,PTHREAD_CREATE_DETACHED);
ret = pthread_create(&tid,&pth_attr,ctl_pthread,(void*)(&ctl_fd));
if(ret < 0)
{
 perror("Create receive pthread error");
 return -1;
}
```

创建聊天室线程，该线程会循环读取fd中服务器发送的信息并打印出来。

```c
struct DataReceiverPara data_receiver_para;
data_receiver_para.socket_fd = afd;
pthread_t receiver_thread_id;
pthread_attr_t attr1;
pthread_attr_init( &attr1 );
pthread_attr_setdetachstate(&attr1,1);
if (pthread_create(&receiver_thread_id, &attr1, data_receive_pthread, (void *)(&data_receiver_para)) == -1){
    error("thread create error\n");
}
```

客户端用多线程收信息，其中收文本信息用基于data_receive_pthread线程函数的线程，收文件信息用基于ctl_pthread线程函数的线程。

```c
static void* data_receive_pthread(void * p){
    struct DataReceiverPara para = *
((struct DataReceiverPara *) p);
    int ret = 0;
    char buf[BUFSIZ];
    for (;;)
    {
        ret = read(para.socket_fd, buf, sizeof(buf));
        write(STDOUT_FILENO, buf, ret);
    }
    pthread_exit(NULL);    //terminate calling thread!
}
void* ctl_pthread(void *p)
{
    pthread_t tid;
    pthread_attr_t pth_attr;
    pthread_attr_init(&pth_attr);
    pthread_attr_setdetachstate(&pth_attr,PTHREAD_CREATE_DETACH
ED);
    int *afd = (int*)p;
    char command[1024];
    while(1)
    {
        memset(command,0,sizeof(command));
        recv(*afd,command,sizeof(command),0);
        if(command[0] == 0)pthread_exit(NULL);
        else pthread_create(&tid,&pth_attr,receive_pthread,NULL
    }
}
```

发送信息部分

先读取初始信息（用户第一步初始化自己的昵称）

```c
char temp[50]; //用户输入的临时存储
char msg[100] = "Init:"; //最终发送的字符串
read_input(temp); //读入temp
strcat(msg, temp); //将temp拼接到msg中
write(afd, msg, strlen(msg)); //向服务器发送msg
safe_flush(stdin); //清除缓冲区
```

用户初始化后进入聊天室，准备进入主循环

主循环前准备

```c
char chatroom_msg[BUFF_SIZE]; //聊天室信息
char filepath[100]; //文件名
```

进入主循环

读取输入流，存入字符串command。将其按冒号分割，将子字符串存入recv_buf中。如果第一个子字符串是Send，则判断为发送文件操作。

```c
if(strcmp(recv_buf[0], "Send") == 0)//Send:Receiver:Filename
{
    if(num == 3){
        if((fd = open(recv_buf[2],O_WRONLY)) <= 0){
            printf("No such file or directory\n");
            continue;
        }
        else close(fd);
        param = (Param*)malloc(sizeof(Param));;
         memset(param->receiver,0,sizeof(param->receiver));
         memset(param->filepath,0,sizeof(param->filepath));
         memset(filepath,0,sizeof(filepath));
         strcpy(param->receiver,recv_buf[1]);
         strcpy(param->filepath,recv_buf[2]);
         strcpy(filepath,recv_buf[2]);
        param->filename = split_filename(filepath);
        memset(command,0,sizeof(command));
        sprintf(command,"%s:%s:%s","Send",param->receiver,param->filename);
        //printf("%s\n",command);
        send(afd,command,sizeof(command),0);
        //printf("%s %s %s\n",param->filepath,param->filename,param->receiver);
        send_file(param);
    }
    else if(num == 2)printf("Please Input Filename\n");
    else if(num == 1)printf("Please Input Receiver and Filename\n");
}
```

否则为正常的聊天信息发送

重置chatroom_msg后，判断用户第一个输入是否Quit（即退出聊天室命令），如果是则跳出主循环并发送退出信息给服务器，如果不是，则将用户输入的内容全部读入chatroom_msg中并以":"结尾（服务器会以冒号为分隔符/终止符读取信息）后发送给服务器。最后不能忘记在每一次循环都要清除输入缓冲区

```c
else{
    memset(chatroom_msg, 0, sizeof(chatroom_msg));
    if (strcmp(recv_buf[0], "quit") == 0 || strcmp(recv_buf[0], uit") == 0)
    {
        strcpy(chatroom_msg, "Quit");
        write(afd, chatroom_msg, strlen(chatroom_msg));
        break;
    }
    strcpy(chatroom_msg, "");
     for (int i = 0; i < num; i++){
         strcat(chatroom_msg, recv_buf[i]);
         strcat(chatroom_msg, ":");
     }
     write(afd, chatroom_msg, strlen(chatroom_msg));
 }
 safe_flush(stdin);
```