学会思考

【思考】这是一个什么项目？远程聊天？缕缕思绪：
1：注册功能;
2：登录功能；
3：远程命令功能；
4：聊天功能
5：心跳功能；

一：两种思维

1：混泥土结构

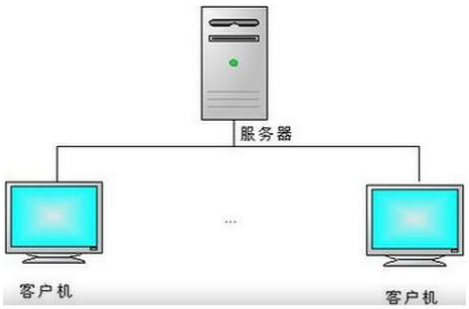
【自下而上，先做模块，注重细节，从难到易】

注册模块→登录模块→远程 shell 命令模块
→聊天模块→心跳模块→组合各个模块

2：框架结构

【自上而下，先搭框架，注重大方向，从易到难】
纵观项目需求(要发送聊天包，命令包，注册包，登录包)
→设计一种协议包
→switch 选择协议包类型(判断客户端是哪种请求)
→读包,写包的过程,包类型与相应模块函数匹配。

二：你需要找感觉！



建立链接

服务器连上客户端，我做过，挺简单！

那就开始吧！Go!

2.1 以此为雏形

搭建服务器，建立与客户端的连接，读写单个字符。

服务器端：

```
int main(void)
{
   int server_sockfd,client_sockfd;
   int server_len,client_len;
   int result;
   struct sockaddr_in server_address;
   struct sockaddr_in client_address;

   unlink("server_socket");
   server_sockfd = socket(AF_INET,SOCK_STREAM,0);

   server_address.sin_family = AF_INET;
   server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
   server_address.sin_port = htons(7683);
   server_len = sizeof(server_address);
   result = bind(server_sockfd,(struct sockaddr *)&server_address,server_len);
   if(result==0)
   {
      puts("bind is success!");
   }else{
      perror("bind error:");
   }
   result = listen(server_sockfd,5);

   if(result==0)
   {
      puts("listen is success!");
   }else{
      perror("listen error:");
   }
   while(1)
  {
    char ch = 'q';
    puts("server waiting\n");
    client_len = sizeof(client_address);
    client_sockfd = accept(server_sockfd,(struct sockaddr *)&client_address,&client_len);
    read(client_sockfd,&ch,1);
    printf("%c from client\n",ch);
    ch++;
    write(client_sockfd,&ch,1);
    printf("send %c to client\n",ch);

    close(client_sockfd);
 }
```

客户端：

```
int main(void)
{
    int sockfd;
    int len;
    struct sockaddr_in address;
    int result;
    char ch = 'A';

    sockfd = socket(AF_INET,SOCK_STREAM,0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port = htons(7683);
    len = sizeof(address);

    result = connect(sockfd,(struct sockaddr *)&address,len);
    if(result == -1)
    {
      perror("oops:client");
      exit(1);
    }
    printf("client send %c to server_socket\n",ch);
    write(sockfd,&ch,1);
    read(sockfd,&ch,1);
    printf("%c from server_socket\n",ch);
    close(sockfd);
    exit(0);
}
```

2.2 进一步思考！

上面的模版，链接建立后，一次只能发送一个字符，之后就关闭套接字了。

那可不行，我需要保证套接字不被关闭，那么我服务器就需要循环的来读写客户端发过来的数据！客户端也需要对应的循环读写！

2.3 不要让建立的连接丢失

所以，服务器端要有这个循环结构：

```
while(1)
{
  char ch = 'q';
  puts("server waiting\n");
  client_len = sizeof(client_address);
  client_sockfd = accept(server_sockfd,(struct sockaddr *)&client_address,&client_len);

  while(1)
  {
    read(client_sockfd,&ch,1);
    printf("%c from client\n",ch);
    ch++;
    write(client_sockfd,&ch,1);
    printf("send %c to client\n",ch);
  }
  close(client_sockfd);
}
```

客户端也要有这个循环结构：

```
result = connect(sockfd,(struct sockaddr *)&address,len);
if(result == -1)
{
  perror("oops:client");
  exit(1);
}
while(1)
{
  printf("client send %c to server_socket\n",ch);
  write(sockfd,&ch,1);
  read(sockfd,&ch,1);
  printf("%c from server_socket\n",ch);
}
close(sockfd);
exit(0);
```

这样改过后，只能发生单个字符！但是，根据项目需要，我们要发生 shell 命令，还要聊天，所以必须发生字符串！

2.4 改造能发送字符串

【怎么思考?】

根据所学知识！

我首先想到的是，客户端要使用 fgets

服务器，要用一个字符数组来接收！

你有没有想到呢？

三：顺便把 shell 命令做了！

当我实现了发生字符串的时候，我好想找到感觉了，因为之前做过 my_shell 作业！我现在手痒了，我想趁热把项目中的远程 shell 实现了！

思考：

服务器收到客户端发过来的 shell 命令

→我需要用 system 函数来执行

→执行的结果我可以通过 dup 映射写到文件里面

→然后把文件里面的内容 read 到一个数组

→然后通过 socket 把数组 write 给客户端！

→完美！！信心大增！

四：心跳功能其实并不可怕

思考：

心跳功能，就是客户端每隔三秒，给服务器发送一个数据包；而服务器会检测客户端的心跳包，如果达到 5 个心跳包没有收到，则表示客户端异常，关闭客户端相关资源！

思路：

怎么每隔三秒发送一个数据包？

之前不是做过每隔三秒打印系统时间的作业吗？

这里就是要使用进程 SIGALARM 信号功能！OK！

五：我好像有把握了！

shell 和心跳都实现了，我突然有了足够的信心！有了一股劲，想干一个通宵写下去！

这就是“情绪猿”的生活？

剩下的主要好想就是聊天了，想到聊天，一定得有两个客户端，两个客户端？但是，我服务器里面 accept 下面是一个 while(1)死循环啊，出不来啊！

完了，好想走不通了！

着急了，抽烟，蹬厕所！

用进程来实现还是用线程来实现？

六：我有点乱了，要睡了

多个客户端链接到一个服务器，那服务器怎么管理这么多客户端呢？我想到线程！

```
        while(1){
                nsock = accept(sockfd,(struct sockaddr*)&clientaddr,(socklen_t*)&len);

                if(nsock == -1){
                        if(errno == EINTR)
                                continue;
                        else
                                break;
                }

                inet_ntop(AF_INET,&clientaddr.sin_addr.s_addr,ipbuf,sizeof(ipbuf));
                printf("client ip:%s connected!\n",ipbuf);
                fprintf(fp,"%s 客户ip:%s连接成功！\n",ipbuf,gettime());
                fflush(fp);

                pthread_create(&id,&attr,read_fun,(void*)nsock);
        }
        close(sockfd);
```

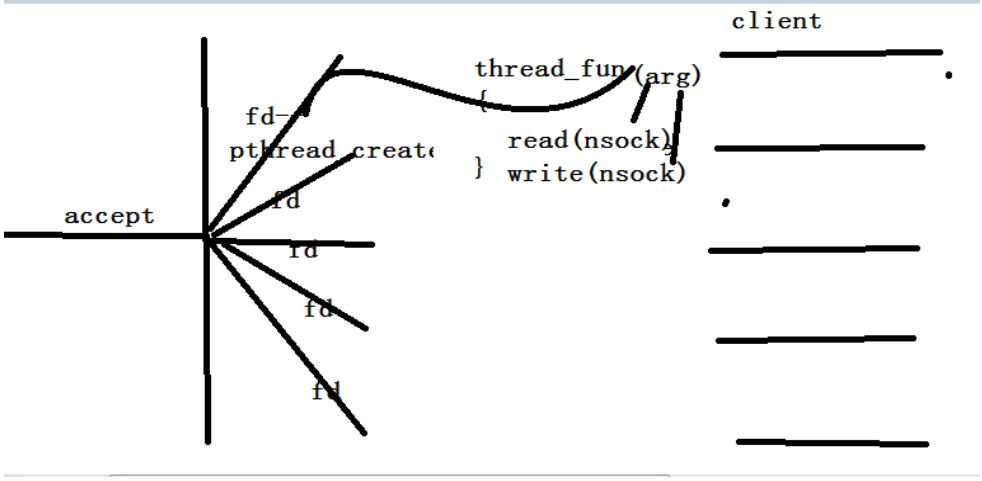
每来一个客户端，就创建一个线程，但是这个线程都是一个函数啊，多个线程可以运行一个程序吗？

```
/*************************
 *线程处理函数
 *参数：void*型的指针
 *返回值：void*型指针
 *************************/
void* read_fun(void* p)
{
//      char buf[1024] = {0};
        int cnt = 0;

        int nsock = (int)p;//将指针转化回int
        Protocol pack = {0};
        pthread_mutex_init(&mutex,NULL);

        while(1)
        {
                printf("fdread=%d\n",nsock);
                cnt = read(nsock,&pack,sizeof(Protocol));//读客户端数据包
                printf("成功读取\n");
                if(0 == cnt)
                {
                        printf("对方网络关闭\n");
                        break;
                }
                else if(-1 == cnt)
                {
                        if(errno == EINTR)
                                continue;
                        else
                        {
                                perror("读取错误");
                                break;
                        }
                }
```

每个客户端的 socket 是一一对应的，这样不会被覆盖吗！
是不是我对 socket 和线程了解的还不够好啊！
原来，我只知皮毛！
我要去问老师了！



线程共享同一段代码，但是他们运行所占的空间是相对独立的啊，所以 socket 不会被覆盖！

我好像已经找到一条光明的路了，我要开始向前挺进了！

七：另一种思维【框架结构】

框架设计！
纵观项目需求，
要发生聊天包，命令包，注册包，登录包！
倒不如先设计一种协议，协议需要什么我就定义什么！不够再补充呗！

```
//--协议包结构体
typedef struct protocol{
        int type;//数据包类型
        char username[16];//用户名
        char userpwd[16];//用户密码
        char chatname[16];//聊天对象用户名
        char data[1024];//数据内容
}Protocol;

#define TYPE_EXIT 0      //退出数据包*
#define TYPE_LOGIN 1     //登录数据包*
#define TYPE_REG 2       //注册数据包*
#define TYPE_MSG 3       //消息数据包
#define TYPE_HEART 4     //心跳数据包*
#define TYPE_CMD 5       //命令数据包*
#define TYPE_OK 6        //正常数据包
#define TYPE_ERR 7       //出错数据包
#define TYPE_CHECK 8     //检查数据包*
#define TYPE_CHAT 9      //聊天数据包*
#define TYPE_READ 10//在线用户数据包
```

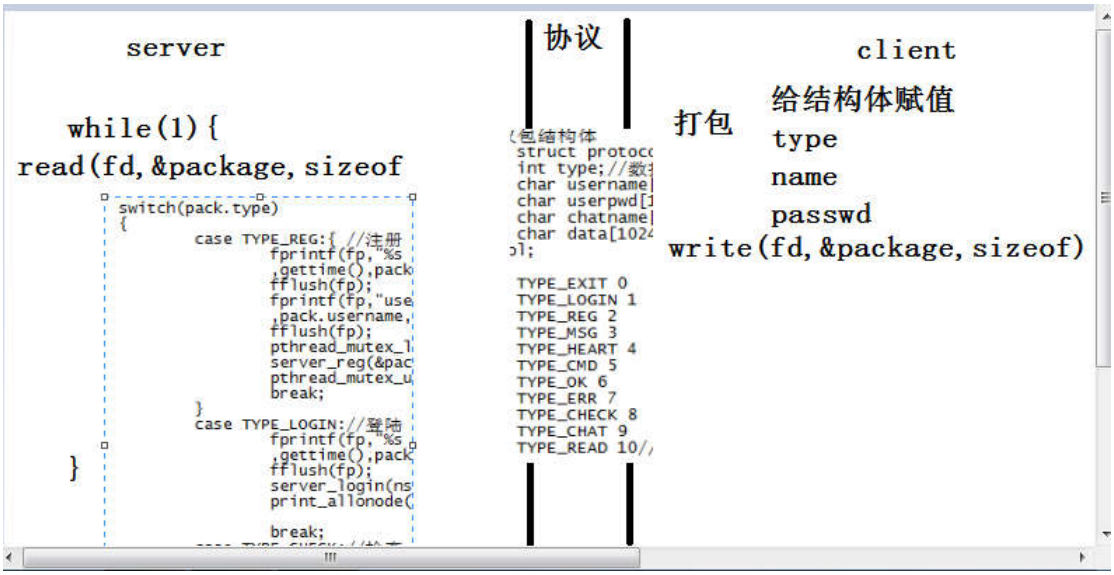
好，服务器和客户端，不管发送什么数据，读通过这个协议包走！

那，既然两边都有各种包的来往，正好可以用上 switch。

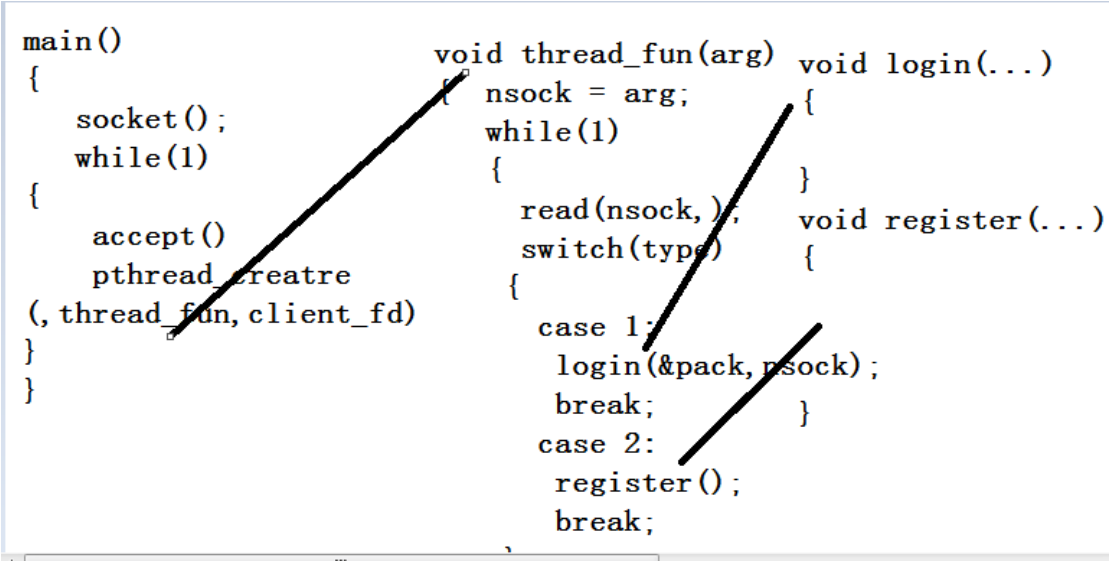
```
switch(pack.type)
{
        case TYPE_REG:{ //注册
                fprintf(fp,"%s client:%s调用了注册\n"
                ,gettime(),pack.data);
                fflush(fp);
                fprintf(fp,"username = %s pwd = %s\n"
                ,pack.username,pack.userpwd);
                fflush(fp);
                pthread_mutex_lock(&mutex);//加互斥锁
                server_reg(&pack);
                pthread_mutex_unlock(&mutex);//解锁
                break;
        }
        case TYPE_LOGIN://登陆
                fprintf(fp,"%s client:%s调用了登录\n"
                ,gettime(),pack.data);
                fflush(fp);
                server_login(nsock,&pack);
                print_allonode();

                break;
        case TYPE_CHECK://检查
                printf("调用了校验\n");
                server_check(nsock,&pack);
                break;
        case TYPE_HEART://心跳
                server_heart(&pack);
                break;
        case TYPE_CMD://命令
                fprintf(fp,"%s 用户：%s调用了shell命令\n"
                ,gettime(),pack.username);
                fflush(fp);
                server_cmd(nsock,&pack);
                break;
        case TYPE_CHAT://聊天
                fprintf(fp,"%s %s要跟%s聊天\n"
                ,gettime(),pack.username,pack.chatname);
                fflush(fp);
                server_chat(nsock,&pack);
                break;
        case TYPE_EXIT://退出
```

我的服务器处于阻塞在 read 函数的状态，所以每当客户端写任何一个类型的包过来，我就可以解析这个包啦！什么类型我就做什么反应，比如这个包是注册包，包含了注册的帐号密码，那我就调用注册函数呗！



对，没错，服务器的循环读写都必须在创建的线程函数里面！



我将要成功了！但是我还要做到数据保存哦,那肯定有链表，链表用来做什么呢？它可以帮我们来保存之前发过的包！

八：我怎么想不到这种设计呢？

看来还是练习不够，做玩这个项目我要利用空余时候再去巩固我之前学过的知识了！

【项目进度安排】

	第一天	第二天	第三天	第四天	第五天	第六天	第七天	
	程序框架的设计	注册	登录	Shell 命令	心跳和心跳异常	聊天功能	完善	

<table>
<tr><td colspan="4">第一天：搭框架</td></tr>
<tr><td>任务</td><td>客户端</td><td>服务器</td><td>容易出现的问题</td></tr>
<tr><td>程序框架的设计，客户端程序结构怎么设计，服务器程序结构怎么设计【关于什么是包，要理解好，服务器和客户端定义好了协议后，就是打包，拆包】</td><td>1：应该有两级菜单，第一级菜单是【注册，登录，退出】，第二级菜单是通过登录进去的【命令，在线用户，聊天，退出】
客户端端的程序框架：程序一运行就应该链接服务器，然后进入主菜单【主菜单应该做成一个循环，因为要考虑到可以重复注册】</td><td>1：服务器的程序框架采用线程，每 accept 一个客户端，创建一个线程，在该线程里面循环 read 数据【一定要打印出返回值和包类型】，然后利用 switch，根据 pack.type 来分包处理，每一种数据包在 case 里面安排一个函数来处理！【这样结构很简洁】</td><td>1：有同学把链接服务器的代码放到了注册函数里面【想一想，如果要连续注册几个用户，难道要链接几次服务器吗】
2：主菜单和子菜单里面都有退出，有同学就直接用 exit 或 return，这是一种野蛮的退出，你要考虑有什么东西要关闭，有什么要释放！
3：客户端和服务器做成两个不同的文件夹，不要混在一起</td></tr>
<tr><td colspan="4">第一天，如果进展慢的 ，不要考虑链表问题，先把框架【客户端和服务器】搭好，并且每一步一定要跟服务器连接调试，首先确信连上了，然后确信进入到了服务器创建的线程函数里面去了，然后输入的用户名和密码确信被服务器接收到！</td></tr>
<tr><td colspan="4">第二天：注册，建立链表</td></tr>
<tr><td>任务</td><td>客户端</td><td>服务器</td><td>容易出现的问题</td></tr>
<tr><td>建立链表【确信注册没问题了，才去做登录】</td><td>链表是只有服务器有的，客户端是没有链表的</td><td>客户端没注册一个用户，服务器就要把这个用户插入到链表里面，注册成功或失败要返回结果给客户端</td><td>1：头结点没有做成全局变量
2：直接操作头结点，没有通过局部变量中转，导致段错误，内存问题
3：每插入一个节点到链表，顺便打印出来，最后要保存到文件，并对文件查看是否保存成功！</td></tr>
<tr><td colspan="4">第二天，加上链表后进行连接调试，再操作链表的加载保存，查重，确保下次注册不能使用之前注册过的帐号！做的快的可以进行登录！</td></tr>
<tr><td colspan="4">第三天：登录</td></tr>
<tr><td>任务</td><td>客户端</td><td>服务器</td><td>容易出现的问题</td></tr>
<tr><td>做登录功能</td><td>使用同一个用户名不</td><td>如果某一个用户登录了，要</td><td>1：用户名已经登</td></tr>
</table>

	能在两个终端同时登录	在链表节点数据里面置一个在线状态，通过这个状态防止同一个用户同时登录	录了应该返回主菜单，如果是密码错误连续三次 2：子菜单退出发送退出包，主菜单退出不需要发生退出包，只需关闭客户端 fd
第三天，登录上用户改为在线状态，对相应的链表也进行操作，如果有疑问，赶紧请教老师。现在你已经开始建立了二级菜单或者三级菜单，你要确保你的 while(1)循环结构的正确性，哪些地方该用 continue 哪些地方用 return 或则 break?			
第四天：远程 shell 命令			
任务	客户端	服务器	容易出现的问题
在线用户查看，命令	每发送一个命令，都要打包发送【即携带包类型，方便服务器根据包类型处理】，子菜单选择了命令后，就进入发送命令函数，并且使用 while（1）	通过 dup，而不要使用 exec 函数，处理命令的函数，不要做成死循环，每处理一个命令就返回	1：客户端 read 的时候接收缓冲区太小导致，有些复杂命令一次存不下 2：有些特殊命令如 rm,cd 不能使用正常 3：错误命令怎么处理
第四天：调试简单的命令是否能实现，确保命令读写不出现阻塞的现象！			
第五天：心跳			
任务	客户端	服务器	容易出现的问题
心跳和心跳异常	发送心态包，通过 SIGALRM	分两部分实现 1：接收到心跳先打印出来 2：然后考虑怎么做心跳异常 【客户端每隔 3 秒发一次心跳包，服务接收之后，登记一下时间（直接用秒数转成 INT）存放到在线链表；服务器每隔 3 秒再去遍历一下在线链表，获取当前时间，减去链表中的时间，判断是否大于 15 秒，进行操作 】	1：发送心跳包在 signal 的捕获函数里面，客户端的 fd 要做成全局 2：检测心跳异常，要关闭客户端 fd，并且删除在线状态
第五天：你的登录注册、shell 命令应该得到保证。再调试心跳程序！			

第六天：聊天			
任务	客户端	服务器	容易出现的问题
聊天	要建立一个线程用于读聊天信息，因为 read 有阻塞功能，如果跟写消息放在一起，就不能连续发生消息	客户端发送聊天信息的时候，会带上聊天对象，服务器通过聊天对象找到它的 fd，然后把信息 write 给他	1:在链表里面没有保存客户端的 fd 2：客户端不知道怎么处理读写问题 3:聊天包应该包含自己的名字还有聊天对象的名字！
第六天：昨晚聊天功能，试着开两到三个客户端进行聊天！确保不出现阻塞！多读或少读的现象！有充沛精力的可以尝试增加新功能！			
第七天：检查、调试、完善			
完善			