

**嵌入式Linux应用程序开发**

**课程设计报告**

**课程设计题目：基于socket和多线程的网络聊天系统的设计**

**专 业：软件工程**

**班 级：1721804**

**姓 名：刘凌杰**

**学 号: 201720180411**

**指导教师:**

**2020年 06 月 20 日**

1. **实验题目**

基于socket和多线程的网络聊天系统的设计。

1. **实验时间、地点**

6月25日,软件楼5楼

1. **实验目的**

使用Linux线程和Socket网络编程接口的基础上，编写网络聊天程序，从而掌握Linux应用程序开发的基本方法，熟悉Linux线程及Socket编程的方法。

1. **实验要求**

使用Linux线程和Socket网络编程接口的基础上，编写网络聊天程序，本课题所设计的系统要求实现以下功能：

* 实现在服务器的控制下，完成多人聊天。
* 在服务器端，使用多线程机制，为多个用户服务。
* 服务器能够对要求接入的客户端进行身份识别，完成聊天数据的转发，并能够将不同用户聊天记录存档。
* 要求写出编译所用到的makefile文件。

**五.实现思路**

(1)服务端实现思路:

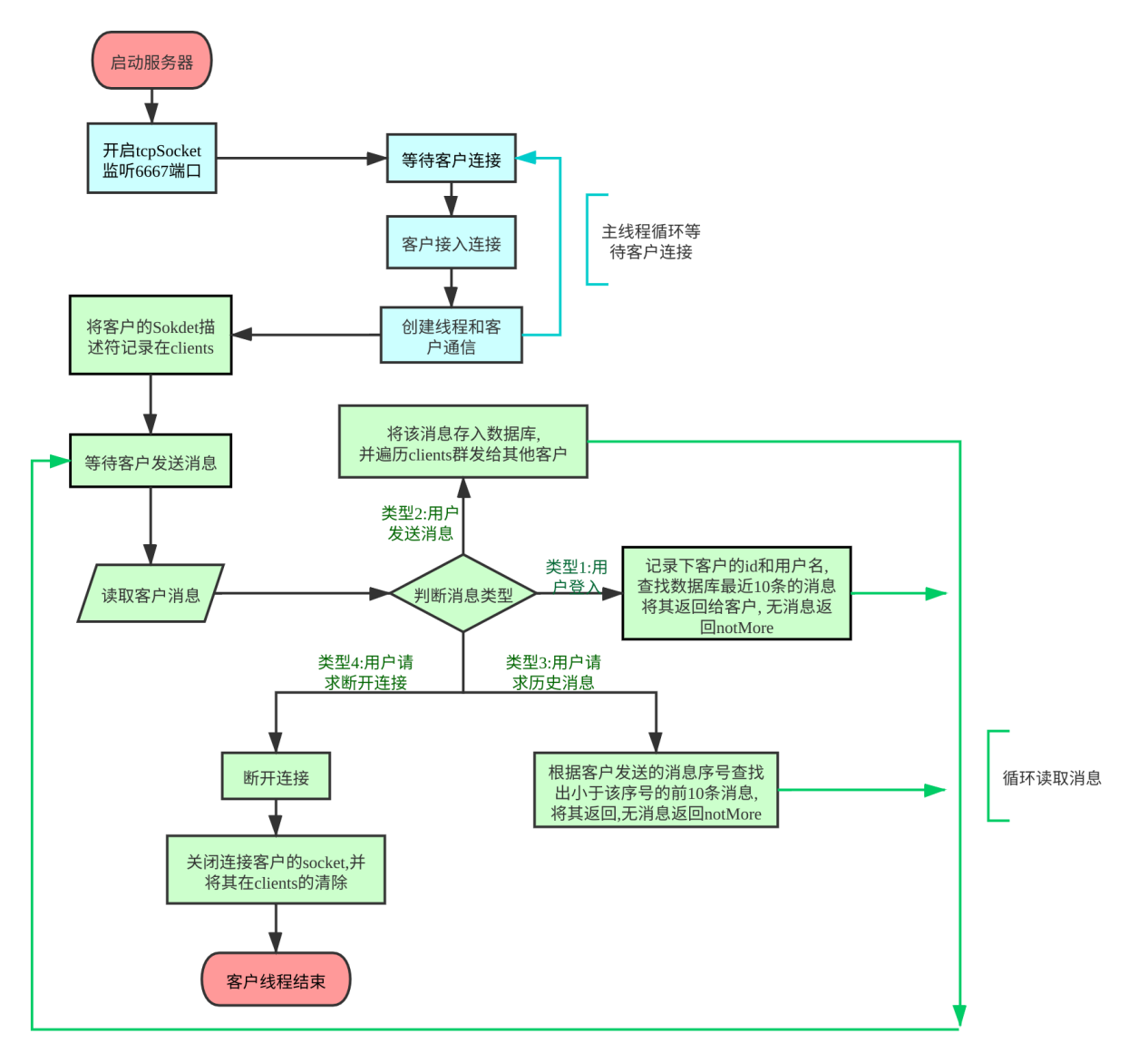


图1-服务端实现流程图

1.启动服务,创建并开启tcp的Socket,循环等待客户接入.

2.客户接入连接,创建一个线程并将接待客户的socket描述符作为参数传递过去.

3.将接待客户的socket描述符保存在全局数组变量clients中.

4.循环读取客户消息

5.读取到客户消息,对客户消息进行类型判断并处理:

(1)类型1:表示客户登入连接,保存用户的名字和id到info数组,并返回最近10条消息,没有返回notMore

(2)类型2:表示客户发送了群聊消息,将该消息保存到数据库并遍利clients群发给其 他在线用户

(3)类型3:表示客户请求历史消息,根据客户发送的消息序号,查询出前10条小于该序号的消息并将其返回,没有返回notMore

(4)类型4:表示客户请求断开连接,将客户的socket关闭,并在clients清除该用户

(2)客户端实现思路

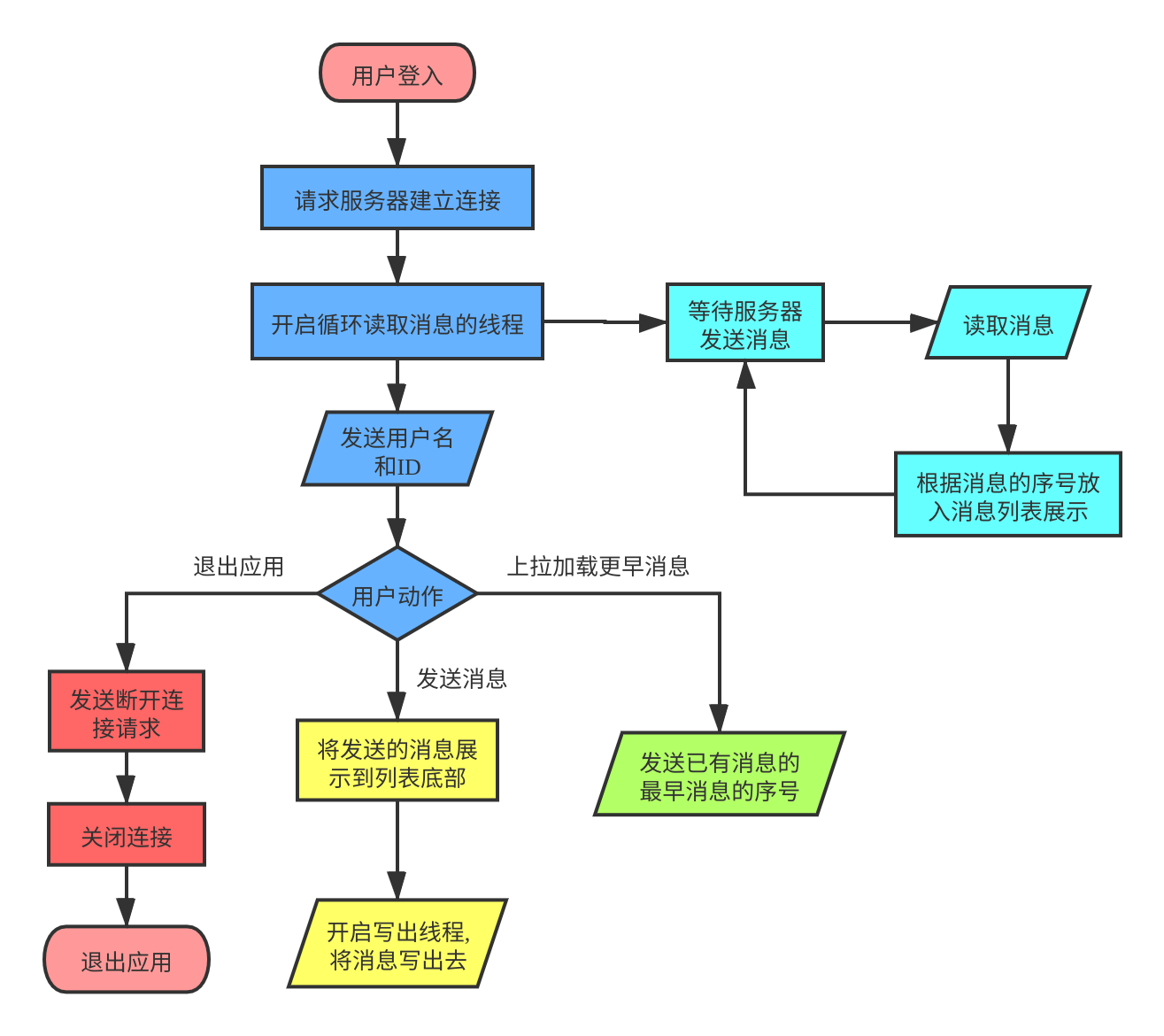


图2-客户端流程图

1.用户启动应用,请求与服务器建立连接.

2.开启循环读取线程,根据消息的序号在消息列表展示消息

3.发送用户名和ID

4.操作:

(1)发送消息:去除危险字符,开启写线程将其发送出去

(2)加载消息:发送已有最早消息的序号,开启写线程将其发送出去

(3)退出应用:开启写线程发送断开连接请求,关闭连接后推出.

1. **实现过程**

**服务端:**

(1)开启监听端口

if((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0){

perror("socket");

return 1;

}

//初始化服务器端的套接字

bzero(&server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET; //设置套接字为ipv4

server\_addr.sin\_port = htons(SERVER\_PORT); //端口转换成网络字节序来设置监听端口

// 代表0.0.0.0，表明所有地址

server\_addr.sin\_addr.s\_addr=inet\_addr("0.0.0.0");

//对于bind，accept之类的函数，里面套接字参数都是需要强制转换成(struct sockaddr \*)

//绑定套接字

if(bind(serverSocket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0){

perror("#绑定套接字失败");

return -1;

}

get\_message\_id();

//设置服务器上的socket为监听状态

if(listen(serverSocket, 5) < 0){

perror("#监听失败");

return -1;

}

(2)初始化消息序号

// 初始化消息序号

void get\_message\_id(){

MYSQL \*conn\_prt;

char result[16]={'\0'};

int num=0;

printf("message\_id初始化中...");

liu\_mysql\_conn(&conn\_prt);

liu\_mysql\_query1(conn\_prt,"select count(\*) from messages",result,&num);

liu\_mysql\_close(conn\_prt);

message\_id=atof(result);

printf("message\_id初始化值 %d\n",message\_id);

}

(3)监听客户接入,客户接入后创建线程与客户通信

while(1){

clientSocket = accept(serverSocket, (struct sockaddr\*)&client\_addr, (socklen\_t\*)&addr\_len);

if(clientSocket < 0){

perror("#创建socket描述符失败");

continue;

}

//表达式：char \*inet\_ntoa (struct in\_addr);inet\_ntoa ip地址转换函数，将网络字节序IP转换为 点分十进制IP

printf("\*\*\*\*^\_^有客户连接啦,IP是 %s,Port是 %d\n", inet\_ntoa(client\_addr.sin\_addr),htons(client\_addr.sin\_port));

//开启客户接待线程

res=pthread\_create(&clients\_thread[number],NULL,thrd\_func,(void\*)clientSocket);

if(res!=0){

printf("#%s 创建线程失败\n",inet\_ntoa(client\_addr.sin\_addr));

exit(res);

}else{

number=(++number)%MAX\_CLIENT;

}

}

close(serverSocket);

(4)在线程中将客户的通信描述符记录在clients\_fd中

pthread\_mutex\_lock(&mutex2); //加锁

for(i=0;i<MAX\_CLIENT;i++){ //存放socket描述符

if(clients\_fd[i]==0){

clients\_fd[i]=fd;

index=i;

break;

}

}

pthread\_mutex\_unlock(&mutex2); //释放锁

(5)循环读取发送的消息,并根据消息类型进行不同处理

while(1){

dataLenth= recv(fd, buffer, 1024, 0); //接受客户的消息放入buffer中

if(dataLenth < 0) {

printf("%d recv null",fd);

continue;

}

buffer[dataLenth] = '\0';

if('4'==buffer[0]){ //客户退出连接

break;

}

//对消息拆分,分为消息类型和和内容

split(buffer,"`",message,&messageLength);

switch (message[0][0]){

case '1':

// split(message[1],"#",info,&len);

login(fd,conn\_prt,message[1],info);

break;

case '2':

send\_message(fd,conn\_prt,message[1],info);

break;

case '3':

more\_message(fd,conn\_prt,message[1]);

break;

default:

}

}

(6)客户登入处理:记录客户信息并发送历史消息

void login(int fd,MYSQL \*conn\_prt,char \*message,char \*\*info){

int num=0,len;

char result[4096]={'\0'};

char \*sql\_str="select \* from messages order by number desc limit 0,10";

split(message,"#",info,&len);

printf("====用户 %s(%s) 上线啦\n",info[1],info[0]);

//发送历史消息

liu\_mysql\_query1(conn\_prt,sql\_str,result,&num);

if(0!=num){

send(fd, result, strlen(result), 0);

} else{

send(fd, "notMore", 7, 0);

}

}

(7)客户发送消息处理:存储消息并群发给在线用户

//转发并存储用户消息

void send\_message(int fd,MYSQL \*conn\_prt,char \*message,char \*\*info){

int len=0,i;

char \*msg[4];

char re\_message[2048]={'\0'};

char sql\_str[2048]={'\0'};

//存储到数据库

split(message,"#",msg,&len);

sprintf(sql\_str,"insert into messages values(%d,'%s','%s','%s','%s','%s')",message\_id,msg[0],msg[1],msg[2],msg[3],msg[4]);

liu\_mysql\_update(conn\_prt,sql\_str); //存储消息

//发送给所有人

sprintf(re\_message,"%d#%s#%s#%s#%s#%s",message\_id,msg[0],msg[1],msg[2],msg[3],msg[4]);

printf("####用户 %s[%s] 发送了消息\n%s\n",info[1],info[0],msg[2]);

for(i=0;i<MAX\_CLIENT;i++){

if(clients\_fd[i]!=fd && clients\_fd[i]!=0){

send(clients\_fd[i], re\_message, strlen(re\_message), 0);

}

}

pthread\_mutex\_lock(&mutex1);

message\_id++; //消息序号+1

pthread\_mutex\_unlock(&mutex1);

}

(8)客户加载消息处理:根据用户发送的消息序号查询出小于该消息序号的前10条消息发送回去

//请求更多消息

void more\_message(int fd,MYSQL \*conn\_prt,char \*message){

int num=0;

char sql\_str[200]={'\0'};

char result1[4096]={'\0'};

sprintf(sql\_str,"select \* from messages where number<%s order by number desc limit 0,10",message);

liu\_mysql\_query1(conn\_prt,sql\_str,result1,&num);

if(0!=num)

send(fd, result1, strlen(result1), 0);

else{

send(fd, "notMore", 7, 0);

}

}

(9)用户请求断开连接后的操作

printf("====用户 %s(%s) 下线\n",info[1],info[0]);

close(fd); //关闭连接

pthread\_mutex\_lock(&mutex2);

clients\_fd[index]=0; //设置为空

pthread\_mutex\_unlock(&mutex2);

liu\_mysql\_close(conn\_prt); //关闭数据库连接

pthread\_exit(NULL);

(10)数据库的连接与操作的封装

//增删改

int liu\_mysql\_update(MYSQL \*conn\_prt,char \*query\_str){

int rc;

rc = mysql\_real\_query(conn\_prt, query\_str, strlen(query\_str)); //执行sql语句

if (0 != rc) {

printf("mysql\_real\_query(): %s\n", mysql\_error(conn\_prt));

return -1;

}

return rc;

}

//初始化mysql句柄,并获取它的地址

int liu\_mysql\_conn(MYSQL \*\*conn\_prt\_prt){

\*conn\_prt\_prt = mysql\_init(NULL); //初始化MYSQL句柄

if(!\*conn\_prt\_prt){

printf("connect mysql failed");

return -1;

}

/\*尝试与mysql数据库连接\*/

if(!mysql\_real\_connect(\*conn\_prt\_prt,"cdb-e3rotq3y.cd.tencentcdb.com","root","LLJ-MySQL","LINUXCHAT",10063,NULL,0)){

printf("failed to coonect mysql:%s\n",mysql\_error(\*conn\_prt\_prt));

}else{

printf("connect success...........\n");

}

return 0;

}

//关闭数据库

void liu\_mysql\_close(MYSQL \*conn\_prt){

mysql\_close(conn\_prt);

}

//查询结果拼接字符串放入result

int liu\_mysql\_query1(MYSQL \*conn\_prt,char \*query\_str,char \*result,int \*num){

MYSQL\_RES \*res = NULL;

MYSQL\_ROW row;

int rc, fields,i,j;

int rows;

// query\_str = "select \* from user";

rc = mysql\_real\_query(conn\_prt, query\_str, strlen(query\_str)); //执行sql语句

if (0 != rc) {

printf("mysql\_real\_query(): %s\n", mysql\_error(conn\_prt));

return -1;

}

res = mysql\_store\_result(conn\_prt); //取出结果集

if (NULL == res) {

printf("mysql\_restore\_result(): %s\n", mysql\_error(conn\_prt));

return -1;

}

rows = mysql\_num\_rows(res); //获取结果集数量

\*num=rows;

fields = mysql\_num\_fields(res); //获取结果集字段数量

memset(result,0,strlen(result));

for(i=0;i<rows;i++){

row = mysql\_fetch\_row(res); //拼接字符串,@为记录分割线,&为字段分割线

if(0!=i)

strcat(result,"@");

for(j=0;j<fields;j++){

if(0!=j)

strcat(result,"#");

strcat(result,row[j]);

}

}

strcat(result,"\0");

mysql\_free\_result(res);

return 0;

}

数据库表(messages):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 是否可以null | 主键 |
| nubmer | 消息序号 | int | 不可以 | 是 |
| name | 用户名 | Varchar(255) | 可以 |  |
| id | 用户ID | Varchar(255) | 可以 |  |
| content | 聊天内容 | Varchat(255) | 可以 |  |
| time | 时间 | datetime | 可以 |  |
| imageUrl | 头像url | Varchar(255) | 可以 |  |

1. makefile文件:

main: main.c mysql.o split.o thread\_client.o

gcc $^ -o $@ -lpthread -lmysqlclient

mysql.o: mysql.c chat.h

gcc -c $< -o $@

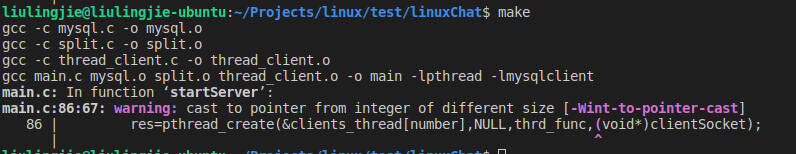
thread\_client.o: thread\_client.c chat.h

gcc -c $< -o $@

split.o: split.c chat.h

gcc -c $< -o $@

编译:



**客户端:**

(1)连接服务器:

//连接服务器

void conn\_server(int \*fd){

struct sockaddr\_in serverAddr; //描述服务器的socket

if((\*fd= socket(AF\_INET, SOCK\_STREAM, 0)) < 0){

perror("socket");

return ;

}

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(6666);

serverAddr.sin\_addr.s\_addr = inet\_addr("106.52.216.126"); //指定服务器端的ip，本地测试：127.0.0.1;inet\_addr()函数，将点分十进制IP转换成网络字节序IP

if(connect(\*fd, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) < 0){ //开始连接

perror("connect fail");

return ;

}

printf("连接到聊天服务器...\n");

}

1. 开启读线程

//开启读线程

res=pthread\_create(&p\_read,NULL,thrd\_func,(void\*)clientSocket);

if(res!=0){

printf("创建读线程失败\n");

exit(res);

}

void \*thrd\_func(void \*arg){

int fd=(long)arg;

int i;

int len=0,rowLen=0,fieldLen=0; //rowLen为消息数量,fieldLen为字段数量

char \*rows[1024]={0}; //记录消息

char \*fields[8]={0}; //记录字段

char \*t[2]={0}; //拆分时间

char rec[10240]={'\0'};

// printf("读线程创建成功\n");

while(1){

rec[0]='\0';

len = recv(fd, rec, 10240, 0);

// printf("%s\n",rec);

rec[len] = '\0';

if(strcmp(rec,"notMore")==0)

continue;

split(rec,"@",rows,&rowLen);

for(i=0;i<rowLen;i++){

split(rows[i],"#",fields,&fieldLen);

split(fields[4]," ",t,&len);

printf("======%s (%s):\n--%s\n",fields[2],t[1],fields[3]);

}

}

}

1. 输入用户名并发送

printf("=========欢迎使用linux多人聊天==========\n");

printf("请输入用户名:\n");

scanf("%s",name);

sprintf(first\_login,"1`noid#%s",name); //发送用户名

send(clientSocket,first\_login,strlen(first\_login),0);

1. 输入消息发送

while(1){

memset(send\_time,0,20);memset(show\_time,0,20);

memset(send\_msg,0,1024);memset(input\_msg,0,1024);

get\_time(send\_time,0);get\_time(show\_time,1); //设置时间

scanf("%s",input\_msg); //输入

if(strcmp(input\_msg,"exit")==0){ //判断是否推出

send(clientSocket,"4",1,0);

break;

}

//将消息发送

sprintf(send\_msg,"2`noid#%s#%s#%s#noimageUrl",name,input\_msg,send\_time);

send(clientSocket,send\_msg,strlen(send\_msg),0);

printf("======%s (%s):\n--%s\n",name,show\_time,input\_msg);

}

close(clientSocket);



图3-运行图

**七.实验总结**

由于程序设计的不够完善,导致程序有以下缺点:线程数组没有自动扩容,需要一次性创建100个,造成了额外开销且无法充分利用,并且也限制了最多只能有100个用户同时在线.并且一条数据包含多个信息是以分隔符将其分开,这也就会出现一个问题,如果数据没有按严格的格式传输,就会导致程序异常,所以要对其进行异常处理.

优点就是逻辑层次清晰,将各个逻辑处理封装在函数中,调用过程简单明了,方便阅读与维护;并且程序运行正常,用户体验良好.

以后在动手开始实现之前,应该要考虑周全并将整个过程先设计出来,而不是盲目的去实现.如要创建哪些变量,要用到那些方法以及多线程情况是否要保证变量的安全访问等等.而这也是是否能够做出优秀软件非常关键的一步.

**八.心得体会**

本次实验用到的知识点较少,socket编程,线程与互斥锁.但是设计与实现对我来说是一大挑战.首先在开始设计之前我分别单独学习并测试了socket编程和线程的启动与关闭,并上网查了如何连接数据库.在做好这些充分准备了便开始了设计.虽然设计思路简单,但是细节方面且要非常的严谨,特别是内存分配与回收,稍有不慎便会出现内存溢出的段错误.经过反复的调试与修复,终于能够稳定运行.

实验工作量不多,但却很考验我的耐心和逻辑.而这锻炼了我的编程能力,技术整合能力,逻辑思维和寻找并修复bug的能力,在写程序时考虑更为周全.也意识到了知识点的欠缺.而且本次实验大部分的错误都是粗心和知识点欠缺导致的,以后我会更加认真学习知识,更加严格要求自己,不再犯这些低级错误.