
Linux 操作系统与程序设计课程设计报告书

基于 Socket 的聊天室设计与实现

组序号：21 组

成员姓名（学号）：

孙晨（159074277）

吴小琮（159084027）

王杰（159074024）

吴琳（159074093）

韩春晖（159074151）

指导老师：郭玉华

提交日期：2018 年 12 月 15 日

《Linux 操作系统与程序设计课程设计》评判分数表

组序号：21 组

日期：2018 年 12 月 15 日

地点：逸夫楼 210

总分数：

答辩成员：孙晨、王杰、吴琳、韩春晖、吴小琮

检查项目类别		分数	备注
程序设计与实现（40 分）	是否多进程/线程：服务器使用多进程，客户端使用 GTK 内部框架，select 以及 gtk 定时中断方式，（已抛弃多线程）		
	功能描述：客户端使用可视化操作界面，各个用户拥有一个主界面和多个二级界面用来与其他客户端进行通讯。		
	基本要求：同步、互斥，通讯，如何解决：同步模式主要通过服务器与数据库之间的关联解决的，互斥模式在数据库中设定 0,1 状态，用户登录互斥，信息传递过程中，服务器生成多进程，每个进程对应一个客户端，解决互斥问题。通讯使用 tcp/ip 协议，严格控制发送与回应，得到回应之后再更新数据库。		
	附加项问题，如何解决：聊天记录使用 lable 进行连接，其次通过数据库进行保存。		
文档（20 分）	字数：合理		
	是否有专业性描述错误：无		
	文档结构（目录，章节编排）是否正确、合理：合理		
答辩（40 分）	基本问题回答情况：		
	附加项问题回答情况：		
	系统演示：		
	调试排错：使用 gdb，打点输出，测试输出等多种方式进行调试。		

教师签字：

日期：

《Linux 操作系统与程序设计课程设计》期中检查情况表

组序号：21 组

填写日期：2018 年 11 月 10 日

填写人（组长）：孙晨

成员（签名）：

开会议题	讨论课程设计的实现中的问题
<p>项目完成情况：</p> <p>（设计，编程，多进程/线程，通讯，服务器，客户端，文档撰写等，总结已经完成什么）</p>	<p>主程序编程基本完成，设计如下：</p> <p>多进程：服务器采用多进程来连接多个服务器，且每个服务器都会新增一个信息处理进程，此进程用来发送信息；</p> <p>多线程：客户端对 server 的监听使用的多线程，但是效果并不理想；通讯无障碍，存在漏接收现象；</p> <p>server 代码能够实现如下功能：用户登录、数据更新、信息发送、信息处理、数据库增删改查等。</p> <p>Client 代码中使用 gtk 框架，存在四级界面：</p> <ol style="list-style-type: none"> 1. 客户端连接服务器，局域网下输入正确的 ip 地址即可连接 2. 在成功登陆之后，输入 server 存在的用户名和密码，并且正确之后，进入三级界面； 3. 聊天界面为用户主界面，当受到发送者信息时，将会产生第四级界面，此界面下能够保存其他用户给自己发来的信息； <p>文档撰写：程序设计基础文档已建立，数据库数据字典，整体框架都已经完善。</p>
<p>目前存在的主要问题及其解决思路</p>	<p>1.中期检测，让本组了解到，单纯靠多线程解决服务器信息，可能会对使信息遗漏，本组会换成主进程接收服务器信息；2.服务器在发送给客户端信息之后不能立刻更新数据库将该条信息设定为已发送，应该等待客户端回应之后再更新；3.第四级界面过于简单，后续会设为多客户多窗口界面。4. 群聊功能有待实现，后续完善。</p>
其它	
准备与老师沟通时间	第六、七、十周周末上课时间
预计答辩时间	12 月 15 日

教师签字：

日期：

《Linux 操作系统与程序设计课程设计》小组会议记录

组序号：21 组

日期：10 月 10 日

地点：逸夫楼 109

参加成员（签名）：

开会议题	确定小组课题内容，分工等
发言记录	<p>课题内容：现要求编写一个基于 Linux 平台 C 语言程序，使用 Linux Socket，多进程或线程等技术编写一个聊天程序。每个客户程序通过服务器发送聊天信息给聊天时的（已经登录连接）所有人或某个人。</p> <p>基本功能：每个客户程序通过服务器发送聊天信息给聊天时的所有人</p> <p>扩展功能：①接收用户输入或显示聊天消息方式为图形化界面；②支持对没有登录的聊天室成员发送聊天记录；③支持查找历史聊天记录；其它方面有如操作方面的速度考虑。</p> <p>初步分工设计：</p> <p style="text-align: center;">架构——孙晨 数据库管理——王杰 JDK 框架——吴琳 信息处理——韩春晖 文本整理——吴小琮</p>
存在的主要问题	<p>1、组内人员之间编程水平有差异，部分组员对于整体框架掌握稍有不足。</p> <p>2、组员线下见面有一定困难，不利于代码整合及总体功能实现。</p> <p>3、对于附加功能选题需讨论。</p> <p>4、现行框架稍有不足，客户端可直接访问服务器，可实现功能但无法运用于实际中。</p>
其它	经过讨论修改后客户可通过访问数据库获取数据，便于多端同时使用本程序。
备注	

教师签字：

日期：

《Linux 操作系统与程序设计课程设计》小组会议记录

组序号：21 组

日期：2018 年 11 月 10 日

地点：逸夫楼 109

参加成员（签名）：

开会议题	讨论课程设计的实现中的问题
发言记录	<p>1、各成员对自己负责部分功能进行阐述，由组长进行问题总汇，相互讨论给出部分解决方案。</p> <p>2、对于改进部分进行讨论。</p> <p>3、部分仍未解决问题由组长带领咨询老师，根据老师意见及建议对程序实现进行改进。</p>
目前存在的主要问题	<p>1、多个窗口同时运行时易崩溃，资源消耗过大</p> <p>2、群聊未能实现。</p> <p>3、发送信息会漏发，数据已存入数据库，但窗口未显示。</p>
其它	<p>咨询过老师后已有初步改进方案。</p> <p>1. 将客户端中多线程处理转换成主进程处理信息；</p> <p>2. 将信息接收机制设为类似于 tcp 协议类型，在服务器收到客户端的回应时再进行数据库更新；</p>
备注	

教师签字：

日期：

1、实验项目目的

《Linux 操作系统课程设计》是一门在课程《Linux 操作系统与程序设计》后独立开设的实验课程。这一门实验课程的开设目的是为了通过学生独立完成一个基于 Linux 平台的较大型应用程序，巩固课堂上学到的 Linux 平台上的编程规范、技术和技巧，培养学生的编写较大型程序的能力和提高学生综合应用素质。

本课程设计实验主要围绕 Linux 平台上主流的基础技术展开，这些技术包括：**Linux 的进程、线程通信和同步技术**；**socket 网络通信技术**等，这些技术可以集中体现并应用在并发程序设计中。通过并发程序的设计与开发，培养学生底层软件开发的能力，并为将来从事 UNIX/Linux 平台开发、嵌入式开发等相对高端的软件开发工作打下基础。

2、实验项目的功能及模块划分

本实验是一个 Linux 下的一个并发程序。基本功能有 server 和 client。在此基础上，

Server 代码能够实现如下功能：

- 1) 用户登录:
 - a) server 将 client 端发送过来的数据流进行拆分获得 id 和 passwd;
 - b) 通过数据库对用户 id 和密码检测;
 - c) 用户 id 唯一登录检测，用户不可重复登陆;
- 2) 信息接收:
 - a) server 将 client 端发送过来的数据流进行拆分获得 sender_id,receiver_id 和 message;
 - b) 获取本地时间作为时间参数;
 - c) 将这四个数据作为一组数据插入到数据库中;
 - d) 将插入数据库的结果存放在 buf 中并发送给客户端;
- 3) 数据库中聊天信息发送:
 - a) server 对 client 的登录检测成功之后，专门为该 client 对应的 socket 分配一个子进程，该进程用来检索数据库中，receiver 与该客户端用户 id 一直且没有发送成功的信息，存放在 result 中;
 - b) 通过固定的流结构，将 sender_id,time 和 msg 信息写入结构中，并发送给服务器;
- 4) 信息确认处理:
 - a) 接收到客户端发送给 server 的信息，如果结构与预先设定的结构一致;
 - b) 通过解析该字符流获得 msg 信息;
 - c) 在数据库中找到该 msg 并将该 msg 的 hasesend 属性设为 1;
- 5) 所有用户 id 获取模块:
 - a) server 将数据库中所有用户 id 获取，并发送给特定的服务器，当用户 id 与之一样，则跳过;
- 6) 服务器用户下线控制模块:

- a) 当收到 client 终止或 exit 信号时，关闭对应的 socket，并将 server 中的用户进行下线更新；

Client 代码中使用 gtk 框架，存在四级界面：

- 1) 客户端连接服务器，局域网下输入正确的 ip 地址即可连接
- 2) 在成功登陆之后，输入 server 存在的用户名和密码，并且正确之后，进入三级界面；
- 3) 聊天界面为用户主界面，当受到发送者信息时，将会产生第四级界面
- 4) 四级界面下能够保存其他用户给自己发来的信息；

拓展功能：

- 1) server 很多模块皆为拓展功能模块
- 2) client 下，使用 gtk3.0 框架
- 3) 使用定时中断模块代替多线程
- 4) 使用多用户多界面模块代替多用户单界面模块
- 5) 使用信息确认模块解决信息漏接问题。

3、设计与实现

3.1 系统结构

3.1.1 整个系统分为客户端和服务端，服务器下包含 sql 数据处理文件和 socket 处理文件，Server 程序主要给 Client 提供基础用户信息和为用户间通讯提供保障，并且保留客户端之间的聊天记录。

其中 Server 结构可以概括如下：

启动服务器（0 级主进程）→监听客户端→生成子进程处理（1 级子进程）→监听该客户端登录→登录成功、子进程再分出一个子进程（2 级子进程）→监听数据库（将未发送给该客户端的信息按一定的时间间隔发送给客户端，此处循环）

1 级子进程→登录成功→给客户端发送其他用户 id→监听客户端给服务器发送的信息→转至信息处理函数（此处循环）{←（直到客户端终止，或者发送 exit 结束监听）}
→更新数据库设定为离线模式

信息处理函数分为三个模块：1.接受客户端发送给其他用户的信息；2.接受客户端发送给服务器的信息确认，3.客户端终止或者退出模块；

3.1.2 客户端，存在四层机构，

（启动客户端）→输入 IP 地址→点击 Connect 按钮链接服务器，登录成功，销毁一级界面，出现 login 界面→输入 id 和 passwd→点击 login 按钮，登录成功，销毁二级界面，生成三级界面（聊天界面）→选择发送用户，输入发送信息→点击 Send 即可发送信息→当有 Server 发来的信息，是 message 则产生四级聊天界面。

3.2 发送进程

3.2.1 基本功能

- a) server 对 client 的登录检测成功之后，专门为该 client 对应的 socket 分配一个子进程，该进程用来检索数据库中，receiver 与该客户端用户 id 一直且没有发送成功的信息，存放在 result 中；
- b) 通过固定的流结构，将 sender_id,time 和 msg 信息写入结构中，并发送给服务器；
- c) server 将数据库中所有用户 id 获取，并发送给 client，当用户 id 与客户端一样，则跳过；

3. 3 接收进程的实现

3. 3. 1. Server 基本功能: 用户登录、信息接收、信息发送、信息确认处理、所有用户 id 获取模块、服务器用户下线控制模块

3. 3. 2. Client 基本功能: 链接服务器、登录、聊天、反馈。

4 数据库访问或文件操作

4. 1. 数据库创建:

```
create table chatuser (  
    id char(9) not null primary key,  
    name varchar(12) not null,  
    passwd char(18) not null,  
    ipaddr char(15) ,  
    port int(5),  
    online binary(1) default 0  
);  
alter table chatuser default character set utf8;  
alter table chatuser modify name varchar(12) character set utf8 not null;  
insert into chatuser( id,name,passwd) values('159074277','孙晨','123456');  
insert into chatuser( id,name,passwd) values('159084027','吴小琮','123456');  
insert into chatuser( id,name,passwd) values('159074024','王杰','123456');  
insert into chatuser( id,name,passwd) values('159074093','159074093','123456');  
insert into chatuser( id,name,passwd) values('159074151','韩春晖','123456');  
create table message (  
    sender char(9) not null,  
    reciver char(9) not null,  
    mes varchar(100) not null,  
    time char(22) not null,  
    hasesend binary(1) default 0  
);  
alter table message default character set utf8;  
alter table message modify mes varchar(100) character set utf8 not null;
```

4. 2 数据库访问, 包括如下 14 个功能模块:

- 1) int getUserId(char userid[10][10]);
- 2) int getOnlineUserId(char onlineuserid[10][10]);
- 3) int getUserName(char username[10][10]);
- 4) int getNoSendMessage(char userid[], struct msgstrcut result[10]);
- 5) int insertMessage(char sender[], char reciver[], char message[], char time[]);
- 6) int updataChatUser(char sender[], char ip[], int port);
- 7) int updataChatUser_offline(char ip[], int port);
- 8) int updataMessageToHavesend(char sender[], char reciver[], char msg[], char time[]);
- 9) char * getUserNameByIp(char ipaddr[], int port);
- 10) char * getPasswdById(const char * userid_text);
- 11) char * getNameById(const char * userid_text);
- 12) int Login(char id[], char passwd[], char ipaddr[], int port);
- 13) int isOnline(char id[]);
- 14) int offOnline();

其他数据库操作方式:

管理员通过终端，对后台数据库进行操作，最高权限。

.....

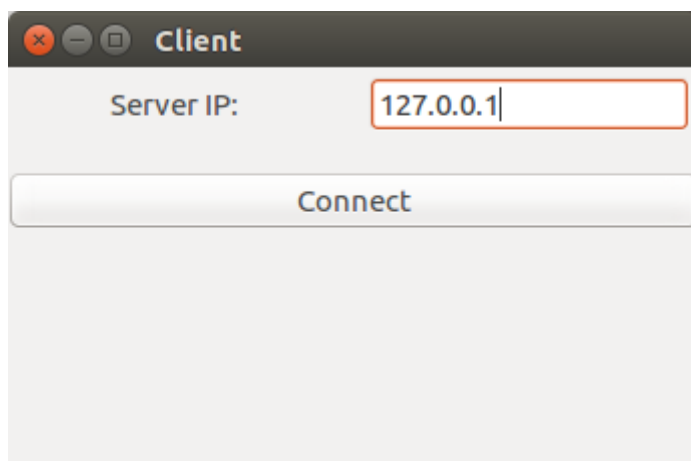
5、通信模块的实现

5.1 通信方式的选择

发送信息需要稳定的通讯方式，这里选择的是 TCP 通讯，以后设定功能为视频语音等实时通讯的系统，需要设定为 UDP 通讯

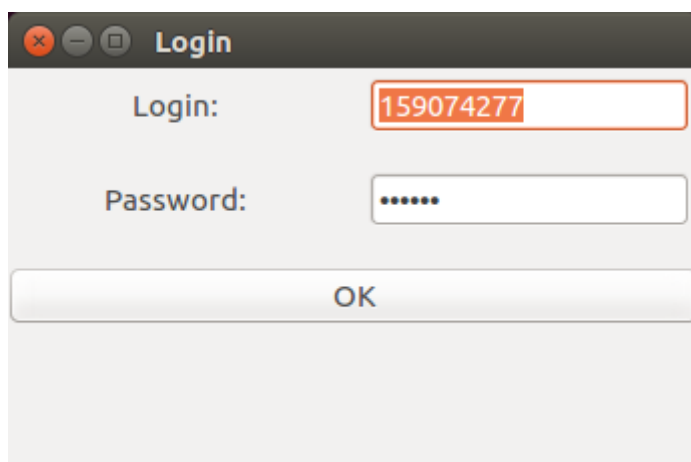
6、测试与调试

6.1 连接服务器：



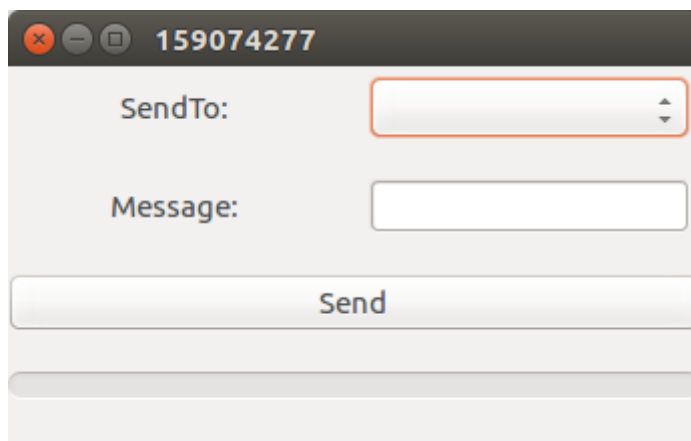
A screenshot of a software window titled "Client". It features a "Server IP:" label followed by a text input field containing "127.0.0.1". Below this is a large, light-gray button labeled "Connect".

6.2.登录：



A screenshot of a software window titled "Login". It contains two input fields: "Login:" with the value "159074277" and "Password:" with masked characters ".....". Below the inputs is a large, light-gray button labeled "OK".

6.3.测试发送数据：



A screenshot of a software window titled "159074277". It has a "SendTo:" label followed by a dropdown menu, and a "Message:" label followed by a text input field. At the bottom is a large, light-gray button labeled "Send".

159074277

SendTo: 159074093

Message: hello

Send

159074277

SendTo: 159074093

Message: hahaha

Send

```
sunc@sunc-HP:~/Desktop/gtk-chat-master (2)/client$ ./client
server ip:127.0.0.1 connect success.
sender login message to server:159074277/123456
159074277 login success
reciver:send message success!
reciver:send message success!
reciver:send message success!
```

sunc@sunc-HP: ~

Your MySQL connection id is 748

Server version: 5.7.24-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use chat

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> select * from chatuser;

id	name	passwd	ipaddr	port	online
159074024	wangjie	123456	127.0.0.1	44856	0
159074093	wuling	123456	127.0.0.1	54914	0
159074151	hanchunhui	123456	127.0.0.1	44600	0
159074277	sunchen	123456	127.0.0.1	51028	1
159084027	wuxiaocong	123456	127.0.0.1	51164	0
169154262	xingying	123456	127.0.0.1	54938	0

6 rows in set (0.00 sec)

mysql> delete from message;

Query OK, 0 rows affected (0.00 sec)

mysql> select * from message;

sender	reciver	mes	time	havesend
159074277	159074093	hello	2018-12-15 13:18:24	0
159074277	159074093	hello	2018-12-15 13:18:26	0
159074277	159074093	hahaha	2018-12-15 13:18:39	0

3 rows in set (0.00 sec)

mysql> █


```
sunc@sunc-HP: ~
+-----+-----+-----+-----+-----+-----+
| 159074277 | 159084027 | hello | 2018-12-15 13:22:15 | 0 | |
| 159074277 | 159084027 | woshishunc | 2018-12-15 13:22:21 | 0 | |
| 159074277 | 159074151 | woshishunc | 2018-12-15 13:22:27 | 1 | |
| 159074277 | 159074024 | woshishunc | 2018-12-15 13:22:32 | 0 | |
| 159074151 | 159074277 | qwewq | 2018-12-15 13:23:56 | 1 | |
| 159074093 | 159074277 | qweqw | 2018-12-15 13:24:48 | 1 | |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

```
mysql> select *select * from message;
```

```
+-----+-----+-----+-----+-----+-----+
| sender | reciver | mes | time | hasesend |
+-----+-----+-----+-----+-----+
| 159074277 | 159074093 | hello | 2018-12-15 13:18:24 | 1 |
| 159074277 | 159074093 | hello | 2018-12-15 13:18:26 | 1 |
| 159074277 | 159074093 | hahaha | 2018-12-15 13:18:39 | 1 |
| 159074277 | 159084027 | hello | 2018-12-15 13:22:15 | 0 |
| 159074277 | 159084027 | woshishunc | 2018-12-15 13:22:21 | 0 |
| 159074277 | 159074151 | woshishunc | 2018-12-15 13:22:27 | 1 |
| 159074277 | 159074024 | woshishunc | 2018-12-15 13:22:32 | 1 |
| 159074151 | 159074277 | qwewq | 2018-12-15 13:23:56 | 1 |
| 159074093 | 159074277 | qweqw | 2018-12-15 13:24:48 | 1 |
| 159074024 | 159074277 | qweeqw | 2018-12-15 13:27:14 | 1 |
| 159074024 | 159084027 | hhh | 2018-12-15 13:27:29 | 0 |
| 159074024 | 159074093 | hhh | 2018-12-15 13:27:36 | 1 |
| 159074024 | 159074093 | hhh | 2018-12-15 13:29:32 | 1 |
| 159074024 | 159074093 | zhenhao | 2018-12-15 13:29:55 | 1 |
| 159074093 | 159074277 | qweqw | 2018-12-15 13:30: 8 | 1 |
| 159074093 | 159074277 | 4kehuduan | 2018-12-15 13:30:18 | 1 |
+-----+-----+-----+-----+-----+-----+
16 rows in set (0.00 sec)
```

```
mysql> select * from chatuser;
```

```
+-----+-----+-----+-----+-----+-----+
| id | name | passwd | ipaddr | port | online |
+-----+-----+-----+-----+-----+-----+
| 159074024 | wangjie | 123456 | 127.0.0.1 | 51038 | 1 |
| 159074093 | wuling | 123456 | 127.0.0.1 | 51034 | 1 |
| 159074151 | hanchunhui | 123456 | 127.0.0.1 | 51036 | 1 |
| 159074277 | sunchen | 123456 | 127.0.0.1 | 51028 | 1 |
| 159084027 | wuxiaocong | 123456 | 127.0.0.1 | 51164 | 0 |
| 169154262 | xingying | 123456 | 127.0.0.1 | 54938 | 0 |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

```
mysql> █
```

sunc@sunc-HP: ~/Desktop/gtk-chat-master (2)/server

```
flag:2
159074024,159074093,zhenhao,2018-12-15 13:29:55
update message set hasend = 1 where sender = "159074024" and reciver = "159074093" and mes =
"zhenhao" and time = "2018-12-15 13:29:55";
get client message:159074277,qweqw
flag:1
time:2018-12-15 13:30: 8qweqw
sockfd :4 sendmessage :159074277,qweqw
ip :127.0.0.1 port : 51034
sender:159074093, reciver:159074277, message:qweqw
insert into message(sender,reciver,mes,time) value("159074093","159074277","qweqw","2018-12-15
13:30: 8");
insert success!
User id is: 159074093
result[0]: sender:159074093, msg:qweqw, time:2018-12-15 13:30: 8.
sender to client 159074277:msg:159074093|2018-12-15 13:30: 8|qweqw
get client message:msg:159074093|2018-12-15 13:30: 8|qweqw
flag:2
159074093,159074277,qweqw,2018-12-15 13:30: 8
update message set hasend = 1 where sender = "159074093" and reciver = "159074277" and mes =
"qweqw" and time = "2018-12-15 13:30: 8";
get client message:159074277,4kehuduan
flag:1
time:2018-12-15 13:30:184kehuduan
sockfd :4 sendmessage :159074277,4kehuduan
ip :127.0.0.1 port : 51034
sender:159074093, reciver:159074277, message:4kehuduan
insert into message(sender,reciver,mes,time) value("159074093","159074277","4kehuduan","2018-1
2-15 13:30:18");
insert success!
User id is: 159074093
result[0]: sender:159074093, msg:4kehuduan, time:2018-12-15 13:30:18.
sender to client 159074277:msg:159074093|2018-12-15 13:30:18|4kehuduan
get client message:msg:159074093|2018-12-15 13:30:18|4kehuduan
flag:2
159074093,159074277,4kehuduan,2018-12-15 13:30:18
update message set hasend = 1 where sender = "159074093" and reciver = "159074277" and mes =
"4kehuduan" and time = "2018-12-15 13:30:18";
get client message:159074024,woshishunc
flag:1
time:2018-12-15 13:32:21woshishunc
sockfd :4 sendmessage :159074024,woshishunc
ip :127.0.0.1 port : 51028
sender:159074277, reciver:159074024, message:woshishunc
insert into message(sender,reciver,mes,time) value("159074277","159074024","woshishunc","2018-
12-15 13:32:21");
insert success!
User id is: 159074277
result[0]: sender:159074277, msg:woshishunc, time:2018-12-15 13:32:21.
sender to client 159074024:msg:159074277|2018-12-15 13:32:21|woshishunc
get client message:msg:159074277|2018-12-15 13:32:21|woshishunc
flag:2
159074277,159074024,woshishunc,2018-12-15 13:32:21
update message set hasend = 1 where sender = "159074277" and reciver = "159074024" and mes =
"woshishunc" and time = "2018-12-15 13:32:21";
```

7、总结

附录：程序代码

```
/*
Copyright (C) 2018 SunChen
*/

#ifndef __COMMON_H
#define __COMMON_H
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h> /*待选*/
#include <errno.h>
#include <fcntl.h> /*for nonblocking*/
#include <netdb.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/epoll.h>
#include <netinet/tcp.h> /*for TCP_XXX defines*/

#define SA struct sockaddr
#define MAXSIZE 128
#define SERV_PORT 9877

#define LISTENQ 5

#define Serv_MAX_EVENTS 1024
#define Client_MAX_EVENTS 2

void setnonblocking(int fd);
void Setsockopt(int sockfd,int level,int optname,const void *optval,socklen_t optlen);
void sock_ntop(const SA *soaddr,char *straddr);
void Listen(int fd,int backlog);
int Socket(int family, int type, int protocol);
void Bind(int fd, const struct sockaddr *sa, socklen_t salen);
int Accept(int lifd,SA * cliaddr,socklen_t * clilen);
int Write(int fd,const char * buf,size_t size);
int Read(int fd,char* buf,size_t size);
int Epoll_create(int size);
void Epoll_ctl(int epfd,int op,int fd,struct epoll_event *ev);
int Epoll_wait(int epfd,struct epoll_event *events,int maxevent,int timeout);
```

```
void str_cli(FILE* fd,int sockfd);
int sock_pton(const char *strIP,SA *soaddr);
#endif //common.h

/*
Copyright (C) 2018 SunChen
*/
#include "common.h"

/*****
/*!

*/
void setnonblocking(int fd)
{

    int flags;

    if((flags=fcntl(fd,F_GETFL,0))<0){
        fprintf(stderr,"fcntl() F_GETFL error :%s\n",strerror(errno));
        exit(1);
    }

    flags |= O_NONBLOCK;

    if(fcntl(fd,F_SETFL,flags)<0){
        fprintf(stderr,"fcntl() F_GETFL error :%s\n",strerror(errno));
        exit(1);
    }

}

/*****
/*!

*/
void Setsockopt(int sockfd,int level,int optname,const void *optval,socklen_t optlen)
{
    if(setsockopt(sockfd,level,optname,optval,optlen)!=0){

        fprintf(stderr,"Setsockopt() error :%s\n",strerror(errno));
        exit(1);
    }
}

/*****/>
```

```
/*!  
  
*/  
int Epoll_wait(int epfd,struct epoll_event *events,int maxevent,int timeout)  
{  
    int nfds;  
  
    while((nfds=epoll_wait(epfd,events,maxevent,timeout))!=-1){  
  
        if(errno==EINTR)  
            continue;  
        else{  
            fprintf(stderr,"epoll_wait() error :%s\n",strerror(errno));  
            exit(1);  
        }  
    }  
  
    return nfds;  
}  
  
/*****  
/*!  
  
*/  
void Epoll_ctl(int epfd,int op,int fd,struct epoll_event *ev){  
  
    if(epoll_ctl(epfd,op,fd,ev)==-1){  
        fprintf(stderr,"fd:%d    epoll_ctl() error:%s\n",fd,strerror(errno));  
        exit(1);  
    }  
    return;  
}  
  
/*****  
/*!  
  
*/  
int Epoll_create(int size){  
  
    int epfd;  
  
    if((epfd=epoll_create(size))!=-1){  
        fprintf(stderr,"epoll_create() error:%s\n",strerror(errno));  
        exit(1);  
    }  
    return epfd;  
}
```

```
/**
 *!

```

```
*/

```

```
int Write(int fd,const char * buf,size_t size){

```

```
    int nleft,nwrite;
    const char *ptr;

```

```
    nleft=size;
    ptr=buf;

```

```
    while(nleft > 0){

```

```
        if((nwrite=write(fd,ptr,nleft)) <= 0){

```

```
            if(errno == EINTR)

```

```
                nwrite = 0;

```

```
            if(errno == EAGAIN){/*针对 ET*/{

```

```
                if(size == nleft)

```

```
                    return -1;

```

```
                else

```

```
                    break;

```

```
            }

```

```
            else{

```

```
                fprintf(stderr,"write() error:%s\n",strerror(errno));

```

```
                exit(1);

```

```
            }

```

```
        }

```

```
        nleft -= nwrite;

```

```
        ptr += nwrite;

```

```
    }

```

```
    return (size-nleft);

```

```
}

```

```
/**
 *!

```

```
*/

```

```
int Read(int fd,char* buf,size_t size){

```

```
    int nread, nleft;

```

```
    char *ptr;

```

```
    ptr=buf;

```

```
    nleft=size;

```

```
    while(nleft > 0){

```

```

        if((nread = read(fd,ptr,nleft))<0){
            if(errno == EINTR)
                nread = 0;
            if(errno == EAGAIN){/*针对 ET*/{
                if(size == nleft)
                    return -1;
                else
                    break;
            }
        }
        else{
            fprintf(stderr,"read() error:%s\n",strerror(errno));
            fflush(stdin);
            fflush(stderr);
            exit(1);
        }
    }
    else if(nread==0)
        break;
    ptr += nread;
    nleft -= nread;
    if(*(ptr-1)=='\n')
        break;
}
return (size-nleft);
}

/*****
/*!

*/
void sock_ntop(const SA *soaddr,char *straddr)
{
    char buf[MAXSIZE],portstr[8];
    switch(soaddr->sa_family){
    case AF_INET:{
        struct sockaddr_in *sin=(struct sockaddr_in *)soaddr;
        if(inet_ntop(AF_INET,&sin->sin_addr,buf,MAXSIZE)==NULL){
            straddr[0]='\0';
            return;
        }
        if(ntohs(sin->sin_port)!=0){
            snprintf(portstr,sizeof(portstr),":%d",ntohs(sin->sin_port));
            strcat(buf,portstr);
        }
        strcpy(straddr,buf);
        return;
    }
}

```

```

}

/*****
/*!

*/

int
Socket(int family, int type, int protocol)
{
    int n;
    if((n=socket(family,type,protocol))<0){
        fprintf(stderr,"socket() error:%s\n",strerror(errno));
        exit(1);
    }
    return(n);
}

/*****
/*!

*/

void Bind(int fd, const struct sockaddr *sa, socklen_t salen)
{
    if(bind(fd,sa,salen)<0){
        fprintf(stderr,"bind() error:%s\n",strerror(errno));
        exit(1);
    }
}

/*****
/*!

*/

void Listen(int fd,int backlog){
    char *ptr;

    if((ptr=getenv("LISTENQ"))!=NULL)
        backlog=atoi(ptr);
    while(listen(fd,backlog)<0){
        if(errno==EINTR)
            continue;
        else{
            fprintf(stderr,"listen() error:%s\n",strerror(errno));
            exit(1);
        }
    }
}

```

```

/*****
/!
 * \brief Accept
 * \param lifd
 * \param cliaddr
 * \param clien
 * \success return connfd,error return -1
 */
int Accept(int lifd,SA * cliaddr,socklen_t *clien)
{
    int connfd;
    while((connfd=accept(lifd,cliaddr,clien)<0)
    {
        if(errno==EINTR)
            continue;
#ifdef EPROTO
        else if(errno==EPROTO || errno==ECONNABORTED)
#else
        else if(errno==ECONNABORTED)
#endif
            continue;
        else{
            fprintf(stderr,"accept() error:%s\n",strerror(errno));
            exit(1);
        }
    }
    return connfd;
}

/*****
/!

 */
void str_cli(FILE * fp,int sockfd)
{
    char buf[MAXSIZE];
    int  n;
    int nfds, epfd;
    struct epoll_event ev, events[Client_MAX_EVENTS];
    int i;

    epfd = Epoll_create( Client_MAX_EVENTS );

    bzero( &ev, sizeof(struct epoll_event) );
    ev.data.fd = fileno(fp);
    ev.events = EPOLLIN; /*LT*/

```

```

Epoll_ctl( epfd, EPOLL_CTL_ADD, fileno(fp), &ev );

bzero( &ev, sizeof( struct epoll_event ) );
ev.data.fd = sockfd;
setnonblocking( sockfd );/**/
ev.events = EPOLLIN|EPOLLET;/**ET*/
Epoll_ctl( epfd, EPOLL_CTL_ADD, sockfd, &ev );
for(;;)
{

    nfds = Epoll_wait(epfd, events, Client_MAX_EVENTS, -1);

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

        if( events[i].data.fd == fileno(fp) )
        {
            //sendMessageWindows(sockfd, "./client51");
            n = Read(fileno(fp), buf, MAXSIZE);
            //printf ("shuru:%s",buf);
            if(n == 0 || strcmp(buf, "exit\n") == 0)
            {
                close(sockfd);
                return;
            }
        }
        else
            Write( sockfd, buf, strlen(buf) );

        bzero(buf, strlen(buf));//test
        fflush(stdin);//test
    }
    if(events[i].data.fd==sockfd)
    {

        /*if n== -1 ,it said that read() return errno==EAGAIN*/
        while((n = Read(sockfd, buf, MAXSIZE)) != -1)
        {

            if(n == 0)
            {
                perror("connect reset!\n");
                close(sockfd);
                return;
            }
            Write(fileno(stderr), buf, strlen(buf));
            bzero(buf, strlen(buf));//test
        }
    }
}

```

```

    }
}

}

}

/*****
/*!
*/
/*success return 1 ,failed return -1*/
int sock_pton(const char* strIP,SA *soaddr){

    int n;

    struct sockaddr_in *sin=(struct sockaddr_in *)soaddr;

    if((n=inet_pton(AF_INET,strIP,&sin->sin_addr))==0){
        perror("IP address is unavaliable!\n");
        return -1;
    }
    else if(n<0){
        fprintf(stderr,"inet_pton() error:%s\n",strerror(errno));
        return -1;
    }
    else
        return 1;

}

```

```

/*
Copyright (C) 2018 SunChen
*/
#ifndef __MYSQLDB_H
#define __MYSQLDB_H

#define SELECT_ID "select id from chatuser;"
#define SELECT_NAME "select name from chatuser;"
#define INSERT "insert into message(sender,reciver,mes,time) value(\"%s\", \"%s\", \"%s\", \"%s\");"
#define UPDATECHATUSER "update chatuser set ipaddr = \"%s\", port = \"%d\",online = 1 where id = \"%s\" and online = 0;"
#define UPDATECHATUSEROFF "update chatuser set online = 0 where ipaddr = \"%s\" and port = \"%d\";"
#define SELECT_NAME_BY_IP "select name from chatuser where ipaddr = \"%s\" and port = \"%d\";"
#define SELECT_PASSWD_BY_ID "select passwd from chatuser where id = %s;"
#define SELECT_NAME_BY_ID "select name from chatuser where id = %s;"

```

```

#define LOGIN "select name from chatuser where id = \"%s\" and passwd = \"%s\";"
#define ONLINE "select id from chatuser where online = 1;"
#define NOSENDMESSAGE "select sender, reciver, mes, time from message where reciver = \"%s\" and
hasesend = 0;"
#define UPDATEMSGHAVESEND "update message set hasesend = 1 where sender = \"%s\" and reciver = \"%s\"
and mes = \"%s\" and time = \"%s\";"
#define ISONLINE "select * from chatuser where id = \"%s\" and online = 1;"
#define OFFLINE "update chatuser set online = 0"

struct msgstrcut
{
    char sender[10];
    char reciver[10];
    char msg[100];
    char time[22];
};

int getUserId(char userid[10][10]);
int getOnlineUserId(char onlineuserid[10][10]);
int getUsername(char username[10][10]);
int getNoSendMessage(char userid[], struct msgstrcut result[10]);
int insertMessage(char sender[], char reciver[], char message[], char time[]);
int updataChatUser(char sender[], char ip[], int port);
int updataChatUser_offline(char ip[], int port);
int updataMessageToHavesend(char sender[], char reciver[], char msg[], char time[]);
char * getUsernameByIp(char ipaddr[], int port);
char * getPasswdById(const char * userid_text);
char * getNameById(const char * userid_text);
int Login(char id[], char passwd[], char ipaddr[], int port);
int isOnline(char id[]);
int offOnline();

#endif

```

```

/*
Copyright (C) 2018 SunChen
*/
#include "mysqldb.h"
#include "common.h"
#include </usr/include/mysql/mysql.h> //我的机器上该文件在/usr/include/mysql 下

/*****
*!
* \brief getUserId
* \return Get all user id list from mysql chat.chatuser
*/

```



```

int getUserId(char userid[10][10])
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集，结构类型
    MYSQL_FIELD *fd ;     //包含字段信息的结构
    MYSQL_ROW row ;       //存放一行查询结果的字符串数组
    int row_i = 0;
    //char qbuf[160];      //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        exit(1);
    }

    if(mysql_query(sock,SELECT_ID))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        //printf("User id is: %s\n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
        strcpy(userid[row_i],(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]));
        ++row_i;
    }
    mysql_free_result(res);
    mysql_close(sock);
    //exit(0);
    return 0;    // 为了兼容大部分的编译器加入此行
}

/*****
/**!
* \brief getOnlineUserId
* \return Get all online user id list from mysql chat.chatuser
*/

int getOnlineUserId(char onlineuserid[10][10])

```

```

{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集，结构类型
    MYSQL_FIELD *fd ;     //包含字段信息的结构
    MYSQL_ROW row ;       //存放一行查询结果的字符串数组
    int row_i = 0;
    //char qbuf[160];      //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        exit(1);
    }

    if(mysql_query(sock,ONLINE))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        printf("User id is: %s\n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
        strcpy(onlineuserid[row_i],(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]));
        ++row_i;
    }
    mysql_free_result(res);
    mysql_close(sock);
    //exit(0);
    return 0;    //. 为了兼容大部分的编译器加入此行
}

/*****
/*!
 * \brief getUsername
 * \return Get all user name list from mysql chat.chatuser
 */
int getUsername(char username[10][10])
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数

```

```

MYSQL_RES *res;          //查询结果集，结构类型
MYSQL_FIELD *fd ;        //包含字段信息的结构
MYSQL_ROW row ;          //存放一行查询结果的字符串数组
int row_i = 0;
//char qbuf[160];        //存放查询 sql 语句字符串

mysql_init(&mysql);
if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
{
    fprintf(stderr,"Couldn't connect to engine!\n%s\n",mysql_error(&mysql));
    perror("");
    exit(1);
}

if(mysql_query(sock,SELECT_NAME))
{
    fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
    exit(1);
}

if (!(res = mysql_store_result(sock)))
{
    fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
    exit(1);
}

while (row = mysql_fetch_row(res))
{
    printf("User id is: %s\n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
    strcpy(username[row_i],(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]));
    ++row_i;
}
mysql_free_result(res);
mysql_close(sock);
//exit(0);
return 0;    // 为了兼容大部分的编译器加入此行
}

/*****
/!
* \brief insertMessage
* \param sender
* \param reciver
* \param message
* \return insert Message into chat.message
*/

int insertMessage(char sender[],char reciver[], char message[], char time[])

```

```

{
    MYSQL mysql,*sock;    //定义数据库连接的句柄， 它被用于几乎所有的 MySQL 函数
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        return 0;
    }
    sprintf(qbuf, INSERT, sender, reciver, message, time);
    printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return 0;
    }
    mysql_close(sock);
    return 1;
}

/*****
/*!

*/

int updataChatUser(char sender[], char ip[], int port)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄， 它被用于几乎所有的 MySQL 函数
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        return -1;
    }
    sprintf(qbuf, UPDATECHATUSER, ip, port, sender);
    printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return -1;
    }
    mysql_close(sock);
    return 1;
}

```

```

/*****
/!
* \brief updataChatUser_offline
* \param ip
* \param port
* \return
*/

```

```

int updataChatUser_offline(char ip[], int port)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n",mysql_error(&mysql));
        perror("");
        return 0;
    }
    sprintf(qbuf, UPDATECHATUSEROFF, ip, port);
    printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return 0;
    }
    mysql_close(sock);
    return 1;
}

```

```

/*****
/*

*/

```

```

char * getUserByNameByIp(char ipaddr[], int port)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;        //查询结果集，结构类型
    MYSQL_FIELD *fd ;      //包含字段信息的结构
    MYSQL_ROW row ;        //存放一行查询结果的字符串数组
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))

```

```

{
    fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
    perror("");
    exit(1);
}

sprintf(qbuf, SELECT_NAME_BY_IP ,ipaddr,port);
if(mysql_query(sock,qbuf))
{
    fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
    exit(1);
}

if (!(res = mysql_store_result(sock)))
{
    fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
    exit(1);
}

while (row = mysql_fetch_row(res))
{
    printf("User name is: %s \n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
    mysql_free_result(res);
    mysql_close(sock);
    return(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]);
}
return "NULL";

}

/*****
/*

*/
char * getPasswdById(const char * userid_text)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集，结构类型
    MYSQL_FIELD *fd ;     //包含字段信息的结构
    MYSQL_ROW row ;       //存放一行查询结果的字符串数组
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
    }

```

```

        exit(1);
    }

    sprintf(qbuf,SELECT_PASSWD_BY_ID,userid_text);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        printf("User passwd is: %s \n",(((row[0]==NULL)&&!strlen(row[0]))) ? "NULL" : row[0]) ;
        mysql_free_result(res);
        mysql_close(sock);
        return(((row[0]==NULL)&&!strlen(row[0]))) ? "NULL" : row[0];
    }
}

/*****
/*

*/
char * getNameById(const char * userid_text)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄， 它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集， 结构类型
    MYSQL_FIELD *fd;       //包含字段信息的结构
    MYSQL_ROW row;         //存放一行查询结果的字符串数组
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        exit(1);
    }

    sprintf(qbuf,SELECT_NAME_BY_ID,userid_text);

```

```

    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        printf("User name is: %s \n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
        mysql_free_result(res);
        mysql_close(sock);
        return(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]);
    }
}

/*****
/*

*/

int Login(char id[], char passwd[] , char ipaddr[],int port)
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄， 它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集， 结构类型
    MYSQL_FIELD *fd ;     //包含字段信息的结构
    MYSQL_ROW row ;       //存放一行查询结果的字符串数组
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n",mysql_error(&mysql));
        return -1;
    }

    sprintf(qbuf, LOGIN ,id, passwd);
    printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return -1;
    }
}

```

```

if (!(res = mysql_store_result(sock)))
{
    fprintf(stderr, "Couldn't get result from %s\n", mysql_error(sock));
    return -1;
}

while (row = mysql_fetch_row(res))
{
    printf("User name is: %s \n", (((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0]));
    if(updateChatUser(id,ipaddr,port) == 0)
    {
        return 0;
    }
    mysql_free_result(res);
    mysql_close(sock);
    return 1;
}
return 0;

}

/*****
/!

*/

int getNoSendMessage(char userid[], struct msgstrcut result[10])
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄， 它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集， 结构类型
    MYSQL_FIELD *fd ;     //包含字段信息的结构
    MYSQL_ROW row ;       //存放一行查询结果的字符串数组
    int row_i = 0;
    char qbuf[160];       //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr, "Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        exit(1);
    }

    sprintf(qbuf, NOSENDMESSAGE ,userid);
    //printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {

```

```

        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        printf("User id is: %s\n",(((row[0]==NULL)&&(!strlen(row[0]))) ? "NULL" : row[0])) ;
        strcpy(result[row_i].sender, row[0]);
        strcpy(result[row_i].reciver,row[1]);
        strcpy(result[row_i].msg,row[2]);
        strcpy(result[row_i].time,row[3]);
        ++row_i;
//        if (row_i >= 10) //超过发送缓存长度，每次检索只能发送十条信息
//        {
//            break;
//        }
    }
    mysql_free_result(res);
    mysql_close(sock);
    return 0;    //. 为了兼容大部分的编译器加入此行

}

/*****
/!!

*/

int updataMessageToHavesend(char sender[], char reciver[], char msg[], char time[])
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        return 0;
    }

    sprintf(qbuf, UPDATEMSGHAVESEND, sender, reciver, msg, time);
    printf ("%s\n",qbuf);

```

```

    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return 0;
    }
    mysql_close(sock);
    return 1;

}

/*****
/

*/

int isOnline(char id[])
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    MYSQL_RES *res;       //查询结果集，结构类型
    MYSQL_ROW row;        //存放一行查询结果的字符串数组
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n\n",mysql_error(&mysql));
        perror("");
        exit(1);
    }

    sprintf(qbuf,ISONLINE,id);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        exit(1);
    }

    if (!(res = mysql_store_result(sock)))
    {
        fprintf(stderr,"Couldn't get result from %s\n", mysql_error(sock));
        exit(1);
    }

    while (row = mysql_fetch_row(res))
    {
        mysql_free_result(res);
        mysql_close(sock);
        return 1 ; //在线
    }
}

```

```

    }
    return 0; //不在线
}

/*****
/!

*/

int offOnline()
{
    MYSQL mysql,*sock;    //定义数据库连接的句柄，它被用于几乎所有的 MySQL 函数
    char qbuf[160];        //存放查询 sql 语句字符串

    mysql_init(&mysql);
    if (!(sock = mysql_real_connect(&mysql,"localhost","root","yushan2d","chat",0,NULL,0)))
    {
        fprintf(stderr,"Couldn't connect to engine!\n%s\n",mysql_error(&mysql));
        perror("");
        return 0;
    }
    sprintf(qbuf, OFFLINE);
    printf ("%s\n",qbuf);
    if(mysql_query(sock,qbuf))
    {
        fprintf(stderr,"Query failed (%s)\n",mysql_error(sock));
        return 0;
    }
    mysql_close(sock);
    return 1;
}

```

```

/*
Copyright (C) 2018 SunChen
*/

#include "common.h"
#include "mysqldb.h"
#include <time.h>

static char userid[10][10] = {"\0"} ;
static char onlineuserid[10][10] = {"\0"} ;
static char username[10][16] = {"\0"} ;

void do_service(int conn,char *addr, int port);
void messageHandle(char *buf,char *addr, int port,int sockfd,char *sender);
void server_login(char *buf,char *loginid,char *passwd,char *sender,char *addr, int port,int sockfd);
void server_reciverMessage(char *buf,char *reciver,char *message,char *sender,char *addr, int port,int sockfd);

```

```

void server_reciverack(char *buf, char *sender);
void server_returnUserId( char *buf, char *loginid, int sockfd);

/*****
/*!
*/

int main(int argc,char *argv[])
{
    signal(SIGCHLD, SIG_IGN);
    int listenfd; //被动套接字(文件描述符)，即只可以 accept, 监听套接字
    struct sockaddr_in peeraddr; //传出参数
    struct sockaddr_in servaddr;
    socklen_t peerlen = sizeof(peeraddr);
    //传入传出参数，必须有初始值
    int conn; // 已连接套接字(变为主动套接字，即可以主动 connect)
    int on = 1;
    pid_t pid;
    getUserId(userid);//更新 ID 表
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    /* servaddr.sin_addr.s_addr = inet_addr("127.0.0.1"); */
    /* inet_aton("127.0.0.1", &servaddr.sin_addr); */

    Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on,sizeof(on));

    Bind(listenfd, (SA *)&servaddr, sizeof(servaddr));

    Listen(listenfd, LISTENQ);
    offOnline();
    while (1)
    {
        conn = Accept(listenfd, (struct sockaddr *)&peeraddr, &peerlen);//3 次握手完成的序列

        printf("sockfd:%d client :ip=%s port=%d connected \n", conn, inet_ntoa(peeraddr.sin_addr),
ntohs(peeraddr.sin_port));
        pid = fork();
        if (pid == -1)
            fprintf(stderr,"fork error :%s\n",strerror(errno));
        else if (pid == 0)
        { // 子进程
            close(listenfd);
            do_service(conn, inet_ntoa(peeraddr.sin_addr), ntohs(peeraddr.sin_port));

```

```

        exit(EXIT_SUCCESS);
    }
    else
        close(conn); //父进程
    }
    return 0;
}

/*****
/*!

*/
void do_service(int conn, char *addr, int port)
{
    int sockfd, n;
    char buf[MAXSIZE];
    sockfd = conn;
    char sender[10] = {"\0"}; //潜在 bug

    while(1)
    {
        //printf("11\n");
        n = read(sockfd, buf, MAXSIZE); //获取客户端发来的信息，存放在 buf 中，后面只需要根据 buf 操
作即可
        printf("get client message:%s\n",buf);
        if(n == 0 || strcmp(buf,"exit") == 0)
        {
            char uname[16] = {"\0"};
            strcpy(uname,getUserByNameByIp(addr, port)); //获取该用户的用户名

            if (updateChatUser_offline(addr, port)) //在数据库中设定该用户离线
            {
                printf("update chatuser( %s set online = 0 ) success!\n",uname);
            }
            else
            {
                printf("update chatuser( %s set online = 0 )failed!\n",uname);
            }

            close(sockfd);
            printf("sockfd :%d disconnected\n",sockfd); //test
            break;
        }
        else //对获取到的信息进行处理
        {
            messageHandle(buf, addr, port, sockfd, sender); //非退出模式，调用服务器信息处理函数
        }
    }
}

```

```

    }
}

/*****
/*!
*/

void messageHandle(char *buf, char *addr, int port, int sockfd, char *sender)
{
    char reciver[10] = {"\0"};
    char message[100] = {"\0"};
    char loginid[10] = {"\0"};
    char passwd[18] = {"\0"};
    int flag = 0; //0 login ,1 sendmessage
    int x = 0;
    while(buf[x] != '\0')
    {
        if (buf[x] == '/') //判断从客户端接收到的信息是否是登录信息
        {
            flag = 0;
            break;
        }
        else if (buf[x] == '~')
        {
            flag = 3;
            break;
        }
        else
            flag = 1;
        ++x;
    }
    if(strncmp(buf,"msg:",4) == 0) //接收信息回应
    {
        flag = 2;
    }

    printf("flag:%d\n",flag);
    if(flag == 0) //登录
    {
        server_login( buf, loginid, passwd, sender, addr, port, sockfd);
    }
    else if(flag == 1) //接受发送信息模块
    {
        server_reciverMessage( buf, reciver ,message, sender, addr, port, sockfd);
    }
    else if(flag == 2) //信息确认模块
    {

```

```

        server_reciverack( buf, sender);
    }
    else if(flag == 3) //给客户端提供用户信息模块
    {
        server_returnUserId( buf, loginid, sockfd);
    }
}

/*****
/!

*/

void server_login(char *buf,char *loginid,char *passwd,char *sender,char *addr, int port,int sockfd)
{
    char * p;
    int i = 0, pid, flag; //flag 返回登录状态
    struct msgstrcut result[10];
    p = buf;
    while(p)//拆分登录信息: id/passwd
    {
        if (*p != '/')
        {
            loginid[i] = *p;
        }
        else
        {
            ++p;
            break;
        }
        ++i;
        ++p;
    }
    strcpy(passwd,p);
    strcpy(sender, loginid);
    printf("buf:%s id:%s passwd:%s\n",buf,loginid,passwd);
    flag = isOnline(loginid); //在线返回 1 不在线返回 0
    if(flag == 0) //不在线状态, 登录
    {
        flag = Login(loginid, passwd, addr, port);
    }
    else
    {
        flag = 0; //在线状态, 不重复登录
    }
    if( flag == 1 )
    {
        strcpy(buf,"success");
    }
}

```



```

printf ("login success\n");
Write(sockfd, buf, 7);

pid = fork();
if (pid == -1)
    fprintf(stderr, "fork error :%s\n", strerror(errno));
else if (pid == 0) //子进程执行信息发送模块
{
    while(1)
    {
        int i = 0;
        memset(result, '\0', sizeof (struct msgstrcut ) * 10);

        getNoSendMessage(loginid, result); //是否有未接受的消息，一次最多存放 10 条未发送的
信息

        while(strlen(result[i].reciver) != 0)
        {
            //设定发送格式  amsg:sender_id|time|msg\0
            printf("result[%d]:          sender:%s,          msg:%s,
time:%s.\n", i, result[i].sender, result[i].msg, result[i].time);
            strcpy(buf, "amsg:");
            strcat(buf, result[i].sender);
            strcat(buf, "|");
            strcat(buf, result[i].time);
            strcat(buf, "|");
            strcat(buf, result[i].msg);
            buf[strlen(buf)] = '\0';
            Write(sockfd, buf, strlen(buf) + 1);
            printf("sender to client %s:%s\n", result[i].reciver, buf);
            ++i;
            sleep(2);
        }
        sleep(2);
    }
}
else //父进程返回上一级
{
    return ;
}
}
else if( flag == 0)
{
    strcpy(buf, "sameid");
    printf ("login failed\n");
    Write(sockfd, buf, 6);
}
else

```

```

    {
        strcpy(buf,"failed");
        printf ("login failed\n");
        Write(sockfd, buf, 6);
    }
    bzero(buf, strlen(buf)); //test

}

/*****
/!

*/

void server_reciverMessage(char *buf,char *reciver,char *message,char *sender,char *addr, int port,int sockfd)
{
    time_t now ;
    struct tm *tm_now ;
    char *p = buf;
    char timebuf[22];
    int i = 0;
    time(&now) ;
    tm_now = localtime(&now) ;
    sprintf(timebuf,"%4d-%2d-%2d %2d:%2d:%2d", \
            tm_now->tm_year+1900, tm_now->tm_mon+1, tm_now->tm_mday, \
            tm_now->tm_hour, tm_now->tm_min, tm_now->tm_sec) ; //获取客户端发送信息的时间，按格式
    设定成字符串参数
    printf("time:%s",timebuf);
    int flag = 0;
    while(p)//msg 信息格式： sender,reciver,mssage
    {
        if (*p != ',')
        {
            reciver[i] = *p;
        }
        else
        {
            ++p;
            break;
        }
        ++i;
        ++p;
    }
    printf ("%s\n",p);
    strcpy(message,p);
    printf("sockfd :%d sendmessage :%s\n", sockfd, buf);//test
    printf("ip :%s port : %d\n", addr, port);//test
    printf("sender:%s, reciver:%s, message:%s\n",sender,reciver,message);
}

```

```

i = 0;
if(strcmp(reciver,sender) == 0)
{
    printf("The reciver cannot be himself\n");
    strcpy(buf,"aThe reciver cannot be himself.\0");
    flag = 2;
}
while( strlen(userid[i] ) != 0 && flag == 0)
{
    if(strcmp(reciver,userid[i]) == 0)
    { //插入数据到 message 数据库中
        //int pid;
        if (insertMessage(sender,reciver,message,timebuf))
        {
            printf("insert success!\n");
            strcpy(buf,"asend message success!\0");
        }
        else
        {
            printf("insert error! sql error\n");
            strcpy(buf,"asql error.\0");
        }
        flag = 1; //判断输入的姓名是否正确
    }
    ++i;
}
if(flag == 0)
{
    printf("insert error! reciver`s id error\n");
    strcpy(buf,"areciver`s id error.\n\0");
}
Write(sockfd, buf, strlen(buf) + 1);
bzero(buf, strlen(buf)); //test
}

void server_reciverack( char *buf,char *sender)
{
    char msg[3][100];
    int x = 4;
    int row = 0, cow = 0;
    memset(msg,'\0',300);
    while(buf[x]) //数据结构: msg:sender|time|msg
    {
        if(buf[x] != '|')
        {
            msg[row][cow] = buf[x];
            ++cow;
        }
    }
}

```

```

    }

    else
    {
        msg[row][cow] = '\0';
        ++row;
        cow = 0;
    }
    ++x;
}
printf("%s,%s,%s,%s\n",msg[0], sender, msg[2], msg[1]);
updateMessageToHaveSend(msg[0], sender, msg[2], msg[1]);
bzero(buf, strlen(buf));//test

}
/*****
/*!

*/
void server_returnUserId( char *buf, char *loginid, int sockfd)
{
    int i = 0;
    bzero(buf, strlen(buf));//test
    //获取所有 id 去除原 id
    while( strlen(userid[i] ) != 0 )
    {
        if(strcmp(loginid,userid[i]) == 0)
        {
            ++i;
            continue;
        }
        else
        {
            if(i == 0)
                strcat(buf,userid[i]);
            else
            {
                strcat(buf,"~");
                strcat(buf,userid[i]);
            }
        }
        ++i;
    }
    strcat(buf,"\0");
    Write(sockfd, buf, strlen(buf) + 1);
    //printf("return buf:%s\n",buf);
    bzero(buf, strlen(buf));//test

```

```
}
```

```
/*
```

```
Copyright (C) 2018 SunChen
```

```
*/
```

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <arpa/inet.h> /*待选*/
```

```
#include <errno.h>
```

```
#include <fcntl.h> /*for nonblocking*/
```

```
#include <netdb.h>
```

```
#include <signal.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <unistd.h>
```

```
#include <sys/wait.h>
```

```
#include <sys/select.h>
```

```
#include <sys/time.h>
```

```
#include <sys/epoll.h>
```

```
#include <netinet/tcp.h> /*for TCP_XXX defines*/
```

```
#define SA struct sockaddr
```

```
#define MAXSIZE 128
```

```
#define SERV_PORT 9877
```

```
#define LISTENQ 5
```

```
#define Serv_MAX_EVENTS 1024
```

```
#define Client_MAX_EVENTS 2
```

```
void sock_ntop(const SA *soaddr, char *straddr);
```

```
void Listen(int fd, int backlog);
```

```
int Socket(int family, int type, int protocol);
```

```
void Bind(int fd, const struct sockaddr *sa, socklen_t salen);
```

```
int Accept(int lifd, SA *cliaddr, socklen_t *clilen);
```

```
int Write(int fd, const char *buf, size_t size);
```

```
int Read(int fd, char* buf, size_t size);
```

```
int Epoll_create(int size);
```

```
void Epoll_ctl(int epfd, int op, int fd, struct epoll_event *ev);
```

```
int Epoll_wait(int epfd, struct epoll_event *events, int maxevent, int timeout);
```

```
void setnonblocking(int fd);
void
Setsockopt(int sockfd,int level,int optname,const void *optval,socklen_t optlen);
```

```
void str_cli(void *argv);
int sock_pton(const char *strIP,SA *soaddr);
```

```
/**
 *!
```

```
 */
void setnonblocking(int fd)
{
    int flags;

    if((flags = fcntl(fd, F_GETFL,0)) < 0)
    {
        fprintf(stderr,"fcntl() F_GETFL error :%s\n", strerror(errno));
        exit(1);
    }

    flags |= O_NONBLOCK;

    if(fcntl(fd,F_SETFL,flags)<0)
    {
        fprintf(stderr,"fcntl() F_GETFL error :%s\n", strerror(errno));
        exit(1);
    }
}
```

```
/**
 *!
```

```
 */
void
Setsockopt(int sockfd,int level,int optname,const void * optval,socklen_t optlen){
    if(setsockopt(sockfd,level,optname,optval,optlen)!=0){

        fprintf(stderr,"Setsockopt error :%s\n",strerror(errno));
        exit(1);
    }
}
```

```

/*****
/*!

*/
int Epoll_wait(int epfd,struct epoll_event *events,int maxevent,int timeout)
{
    int nfd;

    while((nfd=epoll_wait(epfd,events,maxevent,timeout))!=-1){

        if(errno==EINTR)
            continue;
        else{
            fprintf(stderr,"epoll_wait() error :%s\n",strerror(errno));
            exit(1);
        }
    }

    return nfd;
}

/*****
/*!

*/
void Epoll_ctl(int epfd,int op,int fd,struct epoll_event *ev){

    if(epoll_ctl(epfd,op,fd,ev)==-1){
        fprintf(stderr,"fd:%d    epoll_ctl() error:%s\n",fd,strerror(errno));
        exit(1);
    }
    return;
}

/*****
/*!

*/
int Epoll_create(int size){

    int epfd;

    if((epfd=epoll_create(size))!=-1){
        fprintf(stderr,"epoll_create() error:%s\n",strerror(errno));
        exit(1);
    }
}
```

```
        return epfd;
    }

    /**
     *
     */
    int Write(int fd,const char * buf,size_t size){

        int nleft,nwrite;
        const char *ptr;

        nleft=size;
        ptr=buf;

        while(nleft > 0){

            if((nwrite = write(fd, ptr, nleft)) <= 0)
            {
                if(errno == EINTR)
                    nwrite = 0;
                if(errno == EAGAIN)/*针对 ET*/{
                    if(size == nleft)
                        return -1;
                    else
                        break;
                }
            }
            else{
                fprintf(stderr,"write() error:%s\n",strerror(errno));
                exit(1);
            }
            nleft -= nwrite;
            ptr += nwrite;
        }
        return (size-nleft);
    }

    /**
     *
     */
    int Read(int fd,char* buf,size_t size){

        int nread, nleft;
```

```

char *ptr;

ptr=buf;
nleft=size;
printf("a\n");
while(nleft > 0){
    printf("b\n");
    if((nread = read(fd,ptr,nleft))<0){
        printf("c\n");
        if(errno == EINTR)
            nread = 0;
        if(errno == EAGAIN)/*针对 ET*/{
            if(size == nleft)
                return -1;
            else
                break;
        }
        else{
            fprintf(stderr,"read() error:%s\n",strerror(errno));
            fflush(stdin);
            fflush(stderr);
            exit(1);
        }
    }
    else if(nread==0)
        break;
    printf("d\n");
    ptr += nread;
    nleft -= nread;
    if(*(ptr-1)=='\n')
        break;
}

    printf("f\n");
return (size-nleft);
}

int my_read(int fd,void *buffer,int length)
{
int bytes_left;
int bytes_read;
char *ptr;

bytes_left=length;
while(bytes_left>0)
{
    bytes_read=read(fd,ptr,bytes_read);
    if(bytes_read<0)
    {

```

```

        if(errno==EINTR)
            bytes_read=0;
        else
            return(-1);
    }
    else if(bytes_read==0)
        break;
    bytes_left-=bytes_read;
    ptr+=bytes_read;
}
return(length-bytes_left);
}
/*****
/*!

*/

/*success return 1 ,failed return -1*/
int sock_pton(const char* strIP,SA *soaddr){

    int n;
    struct sockaddr_in *sin=(struct sockaddr_in *)soaddr;

    if((n=inet_pton(AF_INET,strIP,&sin->sin_addr))==0){
        perror("IP address is unavaliable!\n");
        return -1;
    }
    else if(n<0){
        fprintf(stderr,"inet_pton() error:%s\n",strerror(errno));
        return -1;
    }
    else
        return 1;

}

```

```

/*
Copyright (C) 2018 SunChen
*/
#include "head.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <string.h>
#include <gtk/gtk.h>
#include <time.h>

```

```
static char userid[10][10] = {"\0"} ;
static char *myid;
static char *mypasswd;
static int sockfd;
```

```
GtkWidget *connectwindow;
GtkWidget *loginwindow;
GtkWidget *sendwindow;
```

```
GtkWidget *status;//ADD
```

```
struct reg_text {
    GtkEntry * id;
    GtkEntry * passwd;
};
```

```
void closeApp(GtkWidget *window, gpointer data)
{
    //printf("Destroy\n");
    gtk_main_quit();
}
```

```
void closeSendWin(GtkWidget *window, gpointer data)
{
    //printf("Destroy\n");
    Write( sockfd, "", 0);
    gtk_main_quit();
}
```

```
void connect_button_clicked(GtkWidget *button, gpointer data)
{
    //int sockfd;
    struct sockaddr_in servaddr;
    int n;

    const char * server_text  = strlen(gtk_entry_get_text(GTK_ENTRY(((GtkWidget *) data))) != 0 ?
        gtk_entry_get_text(GTK_ENTRY(((GtkWidget *) data)) : "127.0.0.1";
    //jadge

    bzero(&servaddr,sizeof(servaddr));

    servaddr.sin_family=AF_INET;
    servaddr.sin_port=htons(SERV_PORT);
    if((n=sock_pton(server_text,(SA *)&servaddr))== -1)
    {
```

```

        printf("wrong formatted ip address.\n");
        return;
    }
    //servaddr.sin_addr.s_addr = inet_addr("192.168.1.101");
    servaddr.sin_addr.s_addr = inet_addr(server_text);
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
        /*这里是模仿作者对 socket()进行简单的包装*/
    {
        printf("socket() error:%s\n",strerror(errno));
        return;
    }

    if(connect(sockfd,(SA *)&servaddr,sizeof(servaddr))<0)
    {
        printf("connect() error:%s\n",strerror(errno));
        close(sockfd);
        return;
    }
    printf("server ip:%s connect success.\n",server_text);
    gtk_widget_destroy (connectwindow);
    return;
}

void login_button_clicked(GtkWidget *button, gpointer data)
{
    struct reg_text * pRt = (struct reg_text *)data;
    char buf[MAXSIZE];
    int n;
    const char * userid_text = strlen(gtk_entry_get_text(GTK_ENTRY(pRt->id))) != 0?
        gtk_entry_get_text(GTK_ENTRY(pRt->id)) : "159074277";
    const char * password_text = strlen(gtk_entry_get_text (GTK_ENTRY(pRt->passwd))) != 0?
        gtk_entry_get_text(GTK_ENTRY(pRt->passwd)) : "123456\0";
    strcpy(buf,userid_text);
    strcat(buf,"/");
    strcat(buf,password_text);
    Write( sockfd, buf, strlen(buf) + 1); //将登录信息发给服务器进行登录
    printf("sender login message to server:%s\n",buf);
    bzero(buf, strlen(buf)); //test
    for(;;)
    {
        while((n = read(sockfd, buf, MAXSIZE)) != -1)
        {
            //printf("%s\n",buf);
            if(n == 0)
            {
                perror("connect reset!\n");
            }
        }
    }
}

```

```

        close(sockfd);
        exit(0);
    }
    else if(strncmp(buf,"sameid",6) == 0)
    {
        perror("login failed! same id\n");
        return;
    }
    else if(strncmp(buf,"failed",6) == 0)
    {
        perror("login failed!\n");
        return;
    }
    else if(strncmp(buf,"success",7) == 0)
    {
        printf ("%s login success\n", userid_text);
        myid = (char *)malloc(10);
        mypasswd = (char *)malloc(18);
        strcpy(myid,userid_text);
        strcpy(mypasswd,password_text);
        bzero(buf, strlen(buf));
        sprintf(buf,"getatherid~%s",myid);
        buf[strlen(buf)] = '\0';
        Write( sockfd, buf, strlen(buf) + 1 );
        //gtk_widget_destroy (loginwindow);
    }
    else if(strlen(buf) == 9 || buf[9] == '~' )
    {

        int x = 0;
        int row = 0, cow = 0;
        while(buf[x])
        {
            if(buf[x] != '~')
            {
                userid[row][cow] = buf[x];
                ++cow;
            }
            else
            {
                userid[row][cow] = '\0';
                ++row;
                cow = 0;
            }
            ++x;
        }
        gtk_widget_destroy (loginwindow);
    }

```

```

        return;

    }
    else if (strncmp(buf,"msg:",4) == 0)
    {
        printf("get server message:%s\n",buf);
    }
    else
        Write(fileno(stderr), buf, n);
    bzero(buf, n); //test
}
}
}

```

```

int connectWindows(int argc, char *argv[])
{
    GtkWidget *server_label;
    GtkWidget *server_entry;
    GtkWidget *ok_button;
    GtkWidget *hbox1;
    GtkWidget *vbox;

    gtk_init(&argc, &argv);
    connectwindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(connectwindow), "Client");
    gtk_window_set_position(GTK_WINDOW(connectwindow), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(connectwindow), 200, 200);

    g_signal_connect(G_OBJECT(connectwindow), "destroy", G_CALLBACK(closeApp), NULL);

    server_label = gtk_label_new("Server IP: ");
    server_entry = gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(server_entry), "127.0.0.1");
    ok_button = gtk_button_new_with_label("Connect");

    g_signal_connect(G_OBJECT(ok_button), "clicked", G_CALLBACK(connect_button_clicked),
        server_entry);

    hbox1 = gtk_hbox_new(TRUE, 5);
    vbox = gtk_vbox_new(FALSE, 10);

    gtk_box_pack_start(GTK_BOX(hbox1), server_label, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(hbox1), server_entry, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox1, FALSE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), ok_button, FALSE, FALSE, 5);
}

```

```
    gtk_container_add(GTK_CONTAINER(connectwindow), vbox);

    gtk_widget_show_all(connectwindow);
    gtk_main();

    return 0;
}

int loginwindows(int argc, char *argv[])
{

    GtkWidget *userid_label, *password_label;
    GtkWidget *userid_entry, *password_entry;
    GtkWidget *ok_button;
    GtkWidget *hbox1, *hbox2;
    GtkWidget *vbox;
    struct reg_text rt;
    //getUserId();
    gtk_init(&argc, &argv);
    loginwindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(loginwindow), "Login");
    gtk_window_set_position(GTK_WINDOW(loginwindow), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(loginwindow), 200, 200);

    g_signal_connect(G_OBJECT(loginwindow), "destroy", G_CALLBACK(closeApp), NULL);

    userid_label = gtk_label_new("Login: ");
    password_label = gtk_label_new("Password: ");
    userid_entry = gtk_entry_new();
    password_entry = gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(userid_entry), "159074277");
    gtk_entry_set_text(GTK_ENTRY(password_entry), "123456");
    gtk_entry_set_visibility(GTK_ENTRY(password_entry), FALSE);
    ok_button = gtk_button_new_with_label("OK");

    g_signal_connect(G_OBJECT(ok_button), "clicked", G_CALLBACK(login_button_clicked),
                    (gpointer)&rt);

    hbox1 = gtk_hbox_new(TRUE, 5);
    hbox2 = gtk_hbox_new(TRUE, 5);
    vbox = gtk_vbox_new(FALSE, 10);

    rt.id = GTK_ENTRY(userid_entry);
    rt.passwd = GTK_ENTRY(password_entry);

    gtk_box_pack_start(GTK_BOX(hbox1), userid_label, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(hbox1), userid_entry, TRUE, FALSE, 5);
```

```

    gtk_box_pack_start(GTK_BOX(hbox2), password_label, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(hbox2), password_entry, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox1, FALSE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox2, FALSE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), ok_button, FALSE, FALSE, 5);
    gtk_container_add(GTK_CONTAINER(loginwindow), vbox);

    gtk_widget_show_all(loginwindow);
    gtk_main();

    return 0;
}

```

```

GtkWidget *sendto_label, *message_label;
GtkWidget *sendto_entry, *message_entry;
GtkWidget *ok_button;
GtkWidget *hbox1, *hbox2;
GtkWidget *vbox;

```

```

#ifdef 1
GtkWidget *frame;
GtkWidget *sends;
GtkWidget *vbox_gui;
GtkWidget *hbox_gui;
GtkWidget *progress;
GtkWidget *fileframe;
GtkWidget *filelabel;
GtkWidget *browse;
GtkWidget *filebox;
#endif

```

```

void send_button_clicked(GtkWidget *button, gpointer data)

```

```

{
    char buf[MAXSIZE];
    int pipes[2];
    int n;

    struct reg_text * pRt = (struct reg_text *)data;
    const char * sendto_text = (char *)
    (*)gtk_combo_box_text_get_active_text(GTK_COMBO_BOX_TEXT(sendto_entry));
    const char * message_text = gtk_entry_get_text(GTK_ENTRY(message_entry));
    if(strlen(message_text) == 0)
    {
        gtk_label_set_text(GTK_LABEL(status), "message can not null.");
        return;
    }
}

```

```

    }
    if (pipe(pipes) == 0)
    {
        write(pipes[1],sendto_text,strlen(sendto_text));
        write(pipes[1],"",1); //写入一个空格，用来分割
        write(pipes[1],message_text,strlen(message_text));
        write(pipes[1],"\\0",1); //写入一个空格，用来分割
        close(0);
        dup(pipes[0]);
    }
    n = read(pipes[0], buf, MAXSIZE);
    Write( sockfd, buf, strlen(buf)+1 );
    return;
}

```

```

char *getfilename()
{

    GtkWidget *dialog;

    dialog = gtk_file_chooser_dialog_new("Select Location",
                                        NULL,
                                        GTK_FILE_CHOOSER_ACTION_SAVE,
                                        GTK_STOCK_SAVE, GTK_RESPONSE_ACCEPT,
                                        NULL);

    if (gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_ACCEPT)
    {
        char *filename;
        filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
        printf("%s", (char *)filename);
        gtk_widget_destroy (dialog);
        return filename;
    }
}

```

```

char Read_SC(int msec)
{
    struct timeval val;
    static int i=0;
    fd_set rset;
    char c;
    int fd=sockfd;

    val.tv_sec = (msec / 1000);
    val.tv_usec = (msec % 1000) * 1000;

```

```

    FD_ZERO(&rset);
    FD_SET(sockfd,&rset);
    switch (select(fd + 1, &rset,0,0,&val))
    {
    case 0: // timeout - treat it like an error.
        return 0;
    case 1: // success - input activity detected.
        printf("111\n");
        read(fd, &c, 1);
        if (c == ' ')
            return 0;
        printf("%c\n",c);
        return c;
    default: // error
        return 0;
    }
    printf("9\n");
}

#define CHATNUM 5
GtkWidget *chatwindow[CHATNUM];
GtkWidget *chatvbox[CHATNUM];
GtkWidget *msg_lable[CHATNUM];
void initchat()
{
    int i ;
    for (i = 0; i < CHATNUM; i++)
    {
        chatwindow[i] = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_position(GTK_WINDOW(chatwindow[i]), GTK_WIN_POS_CENTER);
        gtk_window_set_default_size(GTK_WINDOW(chatwindow[i]), 250, 300);
        msg_lable[i] = gtk_label_new("NULL");
        chatvbox[i] = gtk_vbox_new(FALSE, 10);
        gtk_box_pack_start(GTK_BOX(chatvbox[i]), msg_lable[i], TRUE, TRUE, 5);
        gtk_container_add(GTK_CONTAINER(chatwindow[i]), chatvbox[i]);
    }
}

int chatWindows()
{
    #if 1
        int n = 0, i;
        char buf[MAXSIZE];
        char resendMsg[2000] = {"\0"};
        int isexist = 0;
        static char exist[5][10] = {"\0"}; //保存历史聊天者
        static int existnum = 0; // 保存历史聊天者个数

```

```
while((n = read(sockfd, buf, MAXSIZE)) != -1)
```

```
{
```

```
    printf("reciver:%s\n",buf);
```

```
    if(n == 0)
```

```
    {
```

```
        perror("connect reset!\n");
```

```
        close(sockfd);
```

```
        return 0;
```

```
    }
```

```
    else if(strncmp(buf,"msg:",4) == 0)
```

```
    {
```

```
        //int isexist = 0;
```

```
        char msg[3][100];
```

```
        int x = 4;
```

```
        int row = 0, cow = 0;
```

```
        memset(msg,'\0',300);
```

```
        printf("senderMEssage:%s\n",buf);
```

```
        Write( sockfd, buf, strlen(buf));
```

```
        while(buf[x])
```

```
        {
```

```
            if(buf[x] != '|')
```

```
            {
```

```
                msg[row][cow] = buf[x];
```

```
                ++cow;
```

```
            }
```

```
        else
```

```
        {
```

```
            msg[row][cow] = '\0';
```

```
            ++row;
```

```
            cow = 0;
```

```
        }
```

```
        ++x;
```

```
    }
```

```
    for (i = 0 ;i < existnum; ++i)
```

```
    {
```

```
        if (strcmp(msg[0],exist[i]) == 0 )
```

```
        {
```

```
            strcat(resendMsg,gtk_label_get_text(GTK_LABEL(msg_lable[i]))); //后缀次数过多导致
```

堆栈溢出，这里要进行削减

```
            isexist = 1;
```

```
            break;
```

```
        }
```

```
    }
```

```
    memset(buf,'\0',128);
```

```
    printf("messge:%s/%s/%s\n",msg[0],msg[1],msg[2]);
```

```
    strcat(resendMsg,msg[0]);
```

```

        strcat(resendMsg, " :(");
        strcat(resendMsg, msg[1]);
        strcat(resendMsg, " ) :[ ");
        strcat(resendMsg, msg[2]);
        strcat(resendMsg, " ]\n");
        printf("%s", resendMsg);
        printf("shuchu:%s\n", msg[0]);

    if (isexist == 0)
    { //不存在与该成员的聊天窗口，创建聊天窗口
        char title[30];
        strcpy(title, "s:");
        strcat(title, msg[0]);
        strcat(title, "->");
        strcat(title, myid);
        strcpy(exist[i], msg[0]);
        gtk_window_set_title(GTK_WINDOW(chatwindow[i]), title);
        g_signal_connect(G_OBJECT(chatwindow[i]), "destroy", G_CALLBACK(closeApp),
NULL);

        gtk_label_set_text(GTK_LABEL(msg_lable[i]), resendMsg);
        gtk_widget_show_all(chatwindow[i]);
        ++existnum;
    }
    else
    {
        gtk_label_set_text(GTK_LABEL(msg_lable[i]), resendMsg);
    }

    bzero(buf, strlen(buf)); //test
}
return 1;
}
#endif
}

void choosefile(GtkWidget *browse)
{
    GtkWidget *dialog;
    dialog = gtk_file_chooser_dialog_new("Open
File", GTK_WINDOW(sendwindow), GTK_FILE_CHOOSER_ACTION_OPEN,
GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
GTK_STOCK_OPEN, GTK_RESPONSE_ACCEPT,
NULL);
    if (gtk_dialog_run(GTK_DIALOG (dialog)) == GTK_RESPONSE_ACCEPT)
    {
        char *filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
        gtk_label_set_text(GTK_LABEL(filelabel), filename);
    }
}

```

```
        //strcpy(filename,file);
        gtk_widget_destroy (dialog);
        g_free(filename);
    }
}

int sendfile(GtkWidget *send)
{
#ifdef 0
    char *string=(char *)gtk_combo_box_text_get_active_text(GTK_COMBO_BOX_TEXT(clients));

    if(string==NULL)
        return 0;
    int sockfd=0;
    int i=0;
    while(string[i]!=' ')
    {
        sockfd=sockfd*10+(int)(string[i]-48);
        i++;
    }

#endif

    if (gtk_label_get_text(GTK_LABEL(filelabel))!=NULL)
    {
        char *filename=(char *)gtk_label_get_text(GTK_LABEL(filelabel));
        FILE *ptr;
        char c;
        int size=0;
        int loc=0;
        int d;

        ptr=fopen(filename,"r");
        gtk_progress_bar_set_show_text(GTK_PROGRESS_BAR(progress),1);
        c='0';
        write(sockfd,&c,1);
        gtk_progress_bar_set_text(GTK_PROGRESS_BAR(progress),"Sending File");
        while(fscanf(ptr,"%c",&c)!=EOF)
        {
            d=(int)c;
            write(sockfd,&d,sizeof(int));
        }
        gtk_progress_bar_set_text(GTK_PROGRESS_BAR(progress),"File transfer completed");
        d=-999;
        write(sockfd,&d,sizeof(int));
        fclose(ptr);
        //gtk_widget_destroy(status);
    }
}
```

```
        //g_free (filename);
        return 1;
    }

    //g_free(string);

    return 1;
}

int check(gpointer data)
{
    char c;
    int d;
    int n = 0;
    char buf[MAXSIZE];
    FILE *ptr;
    char *fname;
    //printf("test ");
    if (1)
    {
        c=Read_SC(100);
        //printf("%c\n",c);
        if(c==0)
            return 1;
    }
    else
        return 0;
    switch(c)
    {
    case '0':
        fname=getfilename();
        printf("%s\n",fname);
        printf("Opening file\n");
        //gtk_label_set_text(GTK_LABEL(fstatus),NULL);
        ptr=fopen(fname,"w");
        while(1)
        {

            read(sockfd,&d,sizeof(int));
            //printf("%s",buf);
            if(d==(-999))
            {
                fclose(ptr);
                printf("\nFile closed\n");
                //flag=0;
                free(fname);
                //gtk_label_set_text(GTK_LABEL(fstatus),"File Saved");
                return 1;
            }
        }
    }
```

```

        }
        //read(sock,&c,1);
        printf("%c",d);
        fprintf(ptr,"%c",d);
    }

    //return 1;
case 'l':
    close(sockfd);
    //gtk_widget_destroy_all(window);
    gtk_main_quit();
    exit(0);
case 'a':
default:
    chatWindows(buf);
}
return 1;
printf("over\n");
}

void initialize()
{

    sendwindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(sendwindow), myid);
    gtk_window_set_position(GTK_WINDOW(sendwindow), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(sendwindow), 300, 400);
    sendto_label = gtk_label_new("SendTo: ");
    message_label = gtk_label_new("Message: ");
    sendto_entry = gtk_combo_box_text_new();
    message_entry = gtk_entry_new();
    ok_button = gtk_button_new_with_label("Send");
#ifdef 1
    frame=gtk_frame_new(NULL);
    sends=gtk_button_new_with_label("Send");
    vbox_gui=gtk_vbox_new(1,5);
    hbox_gui=gtk_hbox_new(1,5);
    progress=gtk_progress_bar_new();
    fileframe=gtk_frame_new("Select file");
    filelabel=gtk_label_new(NULL);
    browse=gtk_button_new_with_label("Browse");
    filebox=gtk_hbox_new(0,0);
#endif
    initchat();
}

void signals()

```

```

{
    g_signal_connect(G_OBJECT(sendwindow),"destroy",gtk_main_quit,NULL);
    g_timeout_add_seconds(1,check,NULL);
    g_signal_connect(G_OBJECT(ok_button), "clicked", G_CALLBACK(send_button_clicked),
                     NULL);
    g_signal_connect(browse,"clicked",G_CALLBACK(choosefile),NULL);
    g_signal_connect(sends,"clicked",G_CALLBACK(sendfile),NULL);
    //g_timeout_add(500,check,NULL);

}

void packing()
{
    int i;
    hbox1 = gtk_hbox_new(TRUE, 5);
    hbox2 = gtk_hbox_new(TRUE, 5);
    vbox = gtk_vbox_new(FALSE, 10);
    i = 0 ;
    while( strlen(userid[i]) != 0 ) //将用户信息加载到选择框
    {
        if(strcmp(myid,userid[i]) == 0)
        {
            ++i;
            continue;
        }
        else
            gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(sendto_entry),userid[i]);
        ++i;
    }
    gtk_box_pack_start(GTK_BOX(hbox1), sendto_label, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(hbox1), sendto_entry, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(hbox2), message_label, TRUE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(hbox2), message_entry, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox1, FALSE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox2, FALSE, FALSE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), ok_button, FALSE, FALSE, 5);
    gtk_container_add(GTK_CONTAINER(sendwindow), vbox);
    //gtk_box_pack_start(GTK_BOX(vbox),progress,0,0,5);
    //gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(progress),0);
#ifdef 1
    gtk_box_pack_start(GTK_BOX(vbox_gui),fileframe,0,0,5);
    gtk_container_add(GTK_CONTAINER(fileframe),filebox);
    gtk_box_pack_start(GTK_BOX(filebox),filelabel,0,0,5);
    gtk_box_pack_start(GTK_BOX(filebox),browse,0,0,5);
    gtk_box_pack_start(GTK_BOX(vbox_gui),hbox_gui,1,1,0);
    gtk_box_pack_start(GTK_BOX(hbox_gui),sends,0,1,5);
#endif
}

```

```

        gtk_box_pack_start(GTK_BOX(vbox_gui),progress,0,0,5);
        gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(progress),0);
        gtk_container_add(GTK_CONTAINER(sendwindow),frame);
        gtk_container_add(GTK_CONTAINER(frame),vbox_gui);
        //gtk_progress_bar_set_text(GTK_PROGRESS_BAR(progress),"Disabled");
#endif
        gtk_widget_show_all(sendwindow);

    }
    /**
    *****/
    /*!

    */
int main(int argc,char * argv[])
{
    connectWindows(argc,argv);
    loginwindows(argc,argv);

    //sendMessageWindows(argc,argv);
    initialize();
    signals();
    packing();
    gtk_main();

}
#endif //多线程
/**
*****/
/*!

    */
int sendMessageWindows(int argc, char *argv[])
{
    GtkWidget *sendto_label, *message_label;
    GtkWidget *sendto_entry, *message_entry;
    GtkWidget *ok_button;
    GtkWidget *hbox1, *hbox2;
    GtkWidget *vbox;

    struct reg_text rt;
    int i;
    pthread_t t1;

    gtk_init(&argc, &argv);

```

```

sendwindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(sendwindow), "Communication");
gtk_window_set_position(GTK_WINDOW(sendwindow), GTK_WIN_POS_CENTER);
gtk_window_set_default_size(GTK_WINDOW(sendwindow), 200, 300);

g_signal_connect(G_OBJECT(sendwindow), "destroy", G_CALLBACK(closeSendWin), NULL);

sendto_label = gtk_label_new("SendTo: ");
message_label = gtk_label_new("Message: ");
sendto_entry = gtk_combo_box_text_new();
message_entry = gtk_entry_new();

ok_button = gtk_button_new_with_label("Send");

g_signal_connect(G_OBJECT(ok_button), "clicked", G_CALLBACK(send_button_clicked),
                (gpointer)&rt);

hbox1 = gtk_hbox_new(TRUE, 5);
hbox2 = gtk_hbox_new(TRUE, 5);
vbox = gtk_vbox_new(FALSE, 10);

rt.id = GTK_COMBO_BOX_TEXT(sendto_entry);
rt.passwd = GTK_ENTRY(message_entry);

gtk_box_pack_start(GTK_BOX(hbox1), sendto_label, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox1), sendto_entry, TRUE, TRUE, 5);
gtk_box_pack_start(GTK_BOX(hbox2), message_label, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox2), message_entry, TRUE, TRUE, 5);
gtk_box_pack_start(GTK_BOX(vbox), hbox1, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(vbox), hbox2, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(vbox), ok_button, FALSE, FALSE, 5);
gtk_container_add(GTK_CONTAINER(sendwindow), vbox);

i = 0 ;
while( strlen(userid[i] ) != 0 ) //将用户信息加载到选择框
{
    if(strcmp(myid,userid[i]) == 0)
    {
        ++i;
        continue;
    }
    else
        gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(sendto_entry),userid[i]);
    ++i;
}
//g_timeout_add(100,check,NULL);

```

```

    gtk_widget_show_all(sendwindow);
#endif
    int err = pthread_create(&t1,NULL,str_cli,(void*)NULL);
    if(err!=0)
    {
        printf("thread_create Failed:%s\n",strerror(errno));
    }
    else
    {
        printf("thread_create success\n");
    }
#endif

    gtk_main();

    return 0;

}

/*****
*****/

/*

*/

void str_cli(void *argv)
{
    const int on=1;
    char buf[MAXSIZE];
    int  n;
    int nfd, epfd;
    struct epoll_event ev, events[Client_MAX_EVENTS];
    int i;\n\n\n
        FILE * fp = stdin;
    GtkWidget *chatwindow;
    GtkWidget *vbox;
    GtkWidget *msg_lable;
    char resendMsg[300] = {"\0"};
    chatwindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(chatwindow), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(chatwindow), 250, 300);
    msg_lable = gtk_label_new("NULL");
    vbox = gtk_vbox_new(FALSE, 10);
    gtk_box_pack_start(GTK_BOX(vbox), msg_lable, TRUE, TRUE, 5);
    gtk_container_add(GTK_CONTAINER(chatwindow), vbox);
    //char* exist[5] = {"\0"}; //保存历史聊天者
    //int existnum = 0; // 保存历史聊天者个数
    Setsockopt(sockfd,IPPROTO_TCP,TCP_NODELAY,&on,sizeof(on));
    epfd = Epoll_create( Client_MAX_EVENTS );
    bzero( &ev, sizeof(struct epoll_event) );

```

```

ev.data.fd = fp;
ev.events = EPOLLIN; /*LT*/
Epoll_ctl( epfd, EPOLL_CTL_ADD, fileno(fp), &ev );
bzero( &ev, sizeof( struct epoll_event ) );
ev.data.fd = sockfd;
setnonblocking( sockfd ); /**/
ev.events = EPOLLIN|EPOLLET; /*ET*/
Epoll_ctl( epfd, EPOLL_CTL_ADD, sockfd, &ev );
for(;;)
{

    nfds = Epoll_wait(epfd, events, Client_MAX_EVENTS, -1);

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

        if( events[i].data.fd == fp )
        {

            n = read(fp, buf, MAXSIZE);

            //printf ("shuru:%s\n",buf);
            if(n == 0 || strcmp(buf,"exit\n") == 0)
            {
                close(sockfd);
                return;
            }
            else
            {
                Write( sockfd, buf, strlen(buf)+1 );
            }
            bzero(buf, strlen(buf)); //test
            fflush(0); //test
        }
        if(events[i].data.fd==sockfd)
        {
            /*if n==-1 ,it said that read() return errno==EAGAIN*/
            while((n = Read(sockfd, buf, MAXSIZE)) != -1)
            {
                printf("reciver:%s\n",buf);
                if(n == 0)
                {
                    perror("connect reset!\n");
                    close(sockfd);
                    return;
                }
                else if(strncmp(buf,"msg:",4) == 0)

```

```

    {
        //int isexist = 0;
        char msg[3][100];
        int x = 4;
        int row = 0, cow = 0;
        memset(msg, '\0', 300);
        while(buf[x])
        {
            if(buf[x] != '|')
            {
                msg[row][cow] = buf[x];
                ++cow;
            }

            else
            {
                msg[row][cow] = '\0';
                ++row;
                cow = 0;
            }
            ++x;
        }
        memset(buf, '\0', 128);
        printf("messge: %s/%s/%s\n", msg[0], msg[1], msg[2]);
        strcat(resendMsg, msg[0]);
        strcat(resendMsg, " :(");
        strcat(resendMsg, msg[1]);
        strcat(resendMsg, " :[ ");
        strcat(resendMsg, msg[2]);
        strcat(resendMsg, " ]\n");
        printf("%s", resendMsg);
        // for (i = 0 ; i < existnum; ++i)
        // if (strcmp(msg[0], exist[i] == 0 ))
        // {
        // isexist = 1;
        //
        gtk_label_set_text(GTK_LABEL(msg_lable), resendMsg);
        // break;
        // }
        // if (isexist == 0)
        { //不存在与该成员的聊天窗口，创建聊天窗口
            gtk_window_set_title(GTK_WINDOW(chatwindow), myid);
            g_signal_connect(G_OBJECT(chatwindow), "destroy",
G_CALLBACK(closeApp), NULL);
            gtk_label_set_text(GTK_LABEL(msg_lable), resendMsg);
            gtk_widget_show_all(chatwindow);

```

