功能简述：

包含了一个游戏大厅的服务端和客户端，通过 python ../server/server.py 运行服务端，python ../client/client.py 运行客户端，可以通过修改 ../server/config.py 和 ../client/config.py 来对一些选项进行配置。

客户端配置：

客户端可配置服务器 ip 地址和端口，通过../client/config.py 中的 serverip 和 serverport 来进行配置， 也可以通过在运行客户端时指定参数来进行设置 python ../client/client.py 192.168.187.140 8080

服务器配置：

服务端可配置一下选项：

SERVER_IP	服务器监听地址
SERVER_PORT	服务器监听端口
MAX_LISTEN_NUM	TCP listen 队列容量
MAX_RECV_LEN	TCP 调用 recv 时读取数据长度
SEPRATOR	TCP 消息应用层分包间隔符
SELECT_TIME_OUT	select 调用时的 timeout 间隔，也是整个系统定时器的基础时间
GAME21_PERIOD	21 点游戏触发周期
GAME21_DUARATION	21 点游戏开始后持续可以答题的时间
GAME21_EXPIRES	首次处罚 GAME21 游戏的时间，如果设置为 None 则会选择在服务器启动后首个整点开始进行游戏

支持以下功能：

1. 客户端可以使用账户名，密码登录游戏大厅
   客户端输入 /SIGNIN username password 登录游戏大厅，默认已内置 netease1, netease2 netease3, netease4 几个账户，密码均为 123
2. 注册新用户
   客户端输入 /SIGNUP username password 可以注册新的用户，用户名为 username, 密码为 password
3. 支持聊天
   用户在客户端登录以后，可以直接输入信息进行聊天，聊天信息不得以/开头，输入/开头会被解析为命令，如果是未知命令则会提示用户命令使用方法，用户输入信息后，大厅中的其他用户可以看到消息
4. 账户有在线时长属性
   用户登录以后，可以通过/GETONLINETIME 来获取自己的在线时间，在线时间在用户退出，客户端断开连接，等情况下才会进行更新，如果用户一直保持在线则不会更新在线时间(待改进)
5. 有创建房间功能
   用户登录以后，可以通过 /CREATEROOM roomname 来创建房间，房间的名字为 roomname，用户创建房间后自己会默认加入所创建的房间，其他用户可以通过 /JOINROOM roomname 来加入已创建的房间，在房间中的用户可以通过 /QUITROOM 来退出房间
6. 支持多频道聊天
   用户登录以后，可以通过/W username message 向某个在线用户发送密聊，只有

username 能收到此条消息，其他在线用户无法收到此条消息
用户可以通过/CHATROOM message 来进行房间聊天，此消息只有跟消息发送者处于同一个房间的用户能收到

7. 包含一个 21 点小游戏
大厅会在指定的时间发布 21 点小游戏，随机生成四个 1 到 10 的数，玩家可以通过+.-,*,/和括号让计算的结果尽量接近 21，不能超过 21，超过 21 将会被认为是无效答案，每个人只能回答一次，如果有人回答的结果为 21，此人将获胜，如果到时间了还没有人得出 21 点，则取最大的取胜，若多个人结果相同，则取第一个回答者

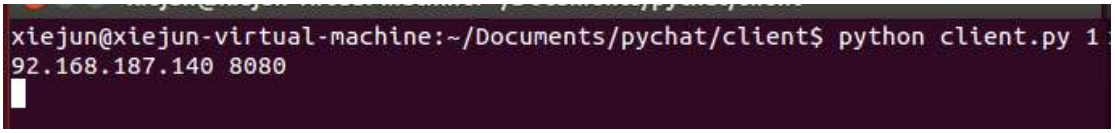
功能测试

1. 启动服务端和客户端

测试在两台 ubuntu 的虚拟机进行，服务器启动在 ip 地址为 192.168.187.140 的机器上，这台机器称为机器 1，客户端运行在 192.168.187.142 的机器上，服务器端口使用 8080，这台机器称为机器 2

1) 首先在机器 1 上进入到服务器代码目录，在第一次运行服务器之前先运行 python user_information.py 会准备好数据库，并建立几个预置账号
2) 运行 python server.py 启动服务器

3) 在机器 2 上进入到客户端代码目录，运行 python client.py 192.168.187.140 8080

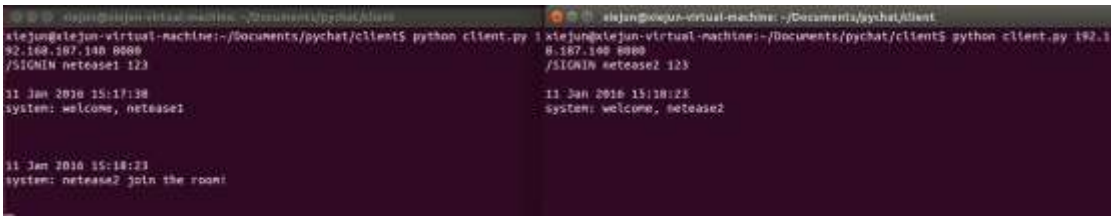


此时服务器和客户端都已经启动

2. 通过账号密码登陆大厅

1) 在客户端输入/SIGNIN netease1 123

在新的 terminal 中启动客户端，使用/SIGNIN netease2 123 登陆第二个用户

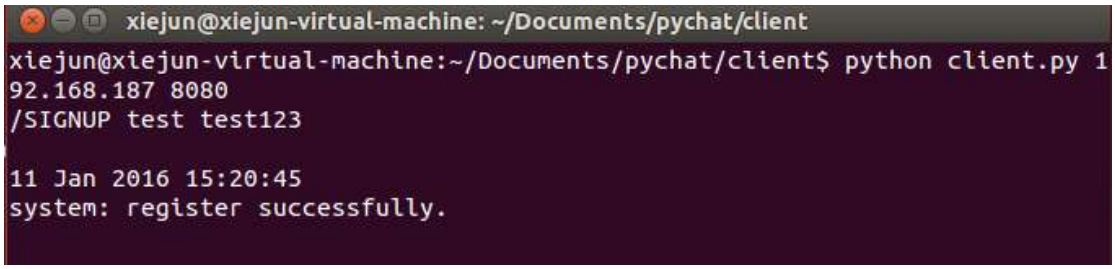


2) 重复登陆， 输入/SIGNIN netease1 123

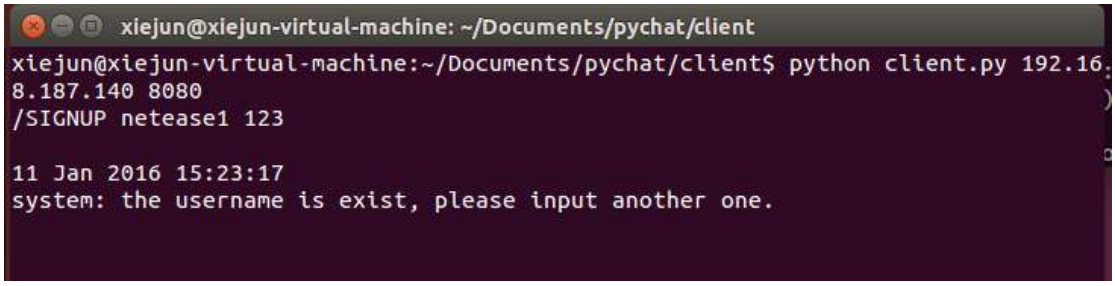
3. 注册新用户

1) 新开一个 terminal 启动客户端，使用/SIGNUP test test123 注册一个用户名为 test 密

码为 test123 的用户

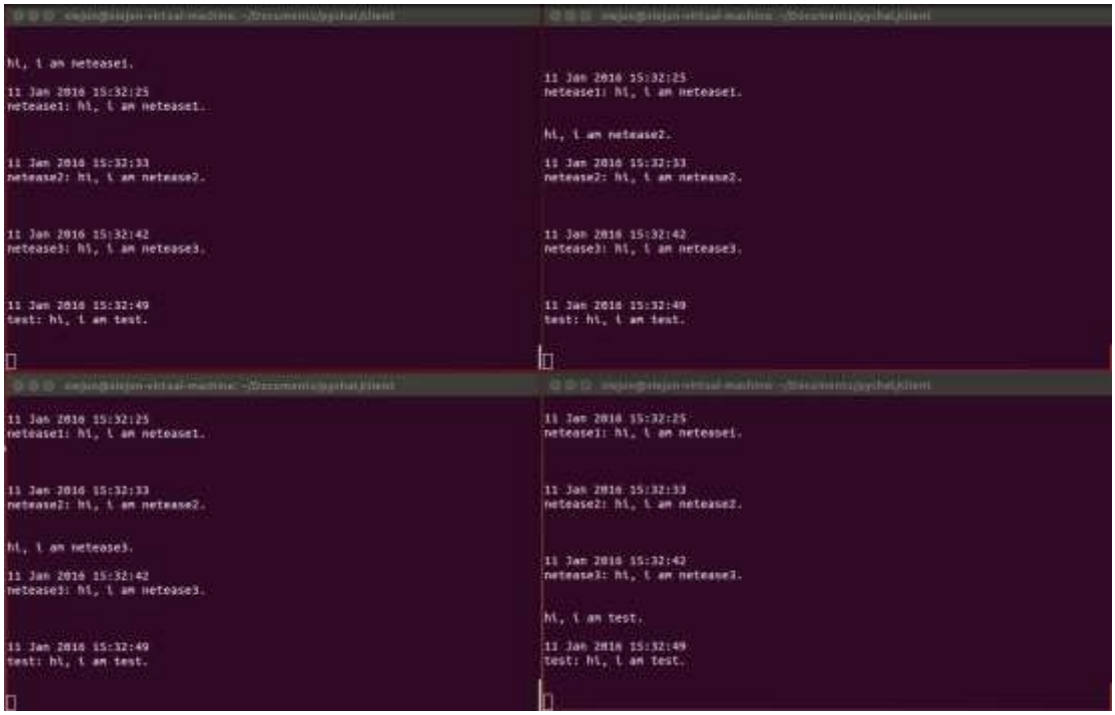


2) 注册用户名已存在，/SIGNUP netease1 123



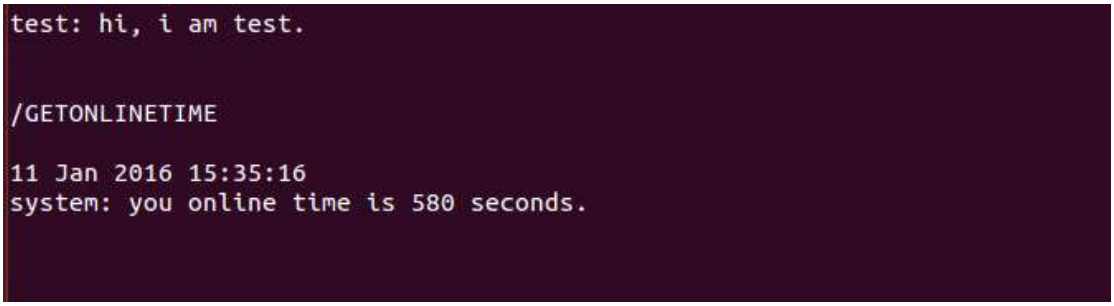
4. 聊天功能测试

登陆 4 个用户 netease1, netease2, netease3, test， 分别在大厅发送自己的 hello + 自己的名字，例如 netease1 输入 hello, I am netease1.



5. 在线时长测试

用户输入/GETONLINETIME 获取自己的在线时间

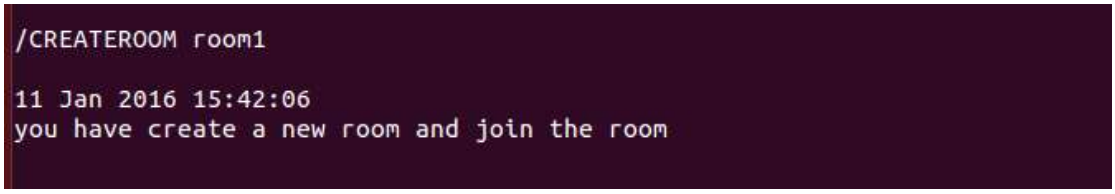


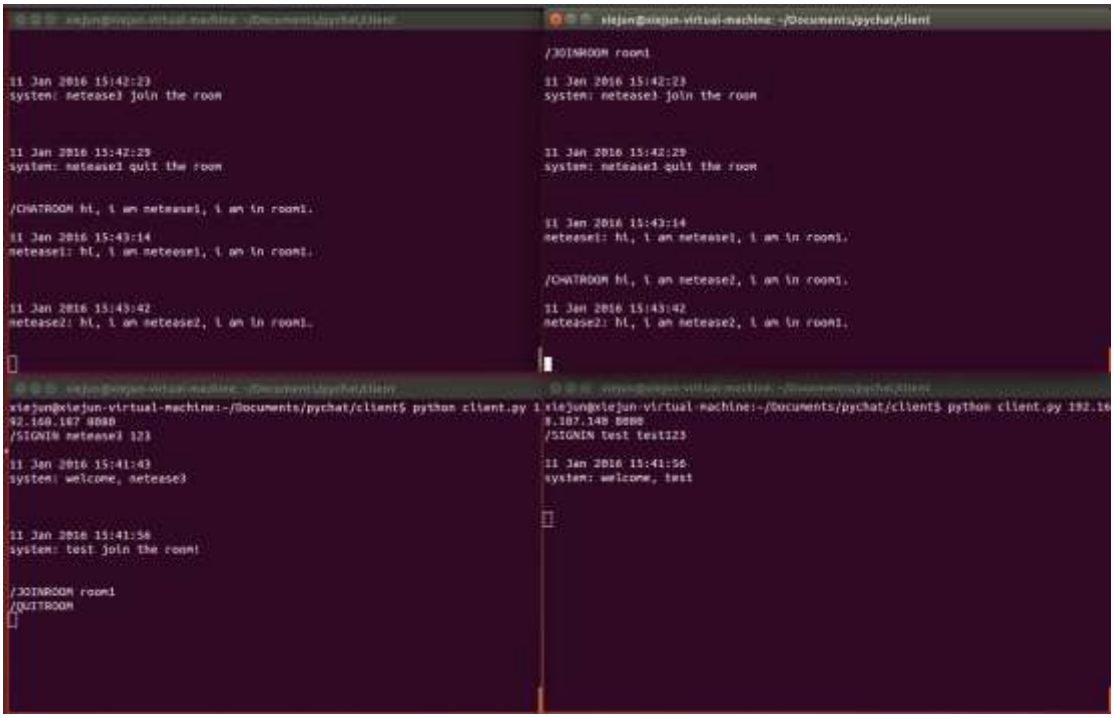
6. 创建房间、加入房间、退出房间功能

netease1 创建一个名为 room1 的房间，netease2 加入此房间， netease3 加入此房间，netease3 退出此房间， 房间中剩余的 netease1， netease2 朝房间内发送消息

步骤如下：

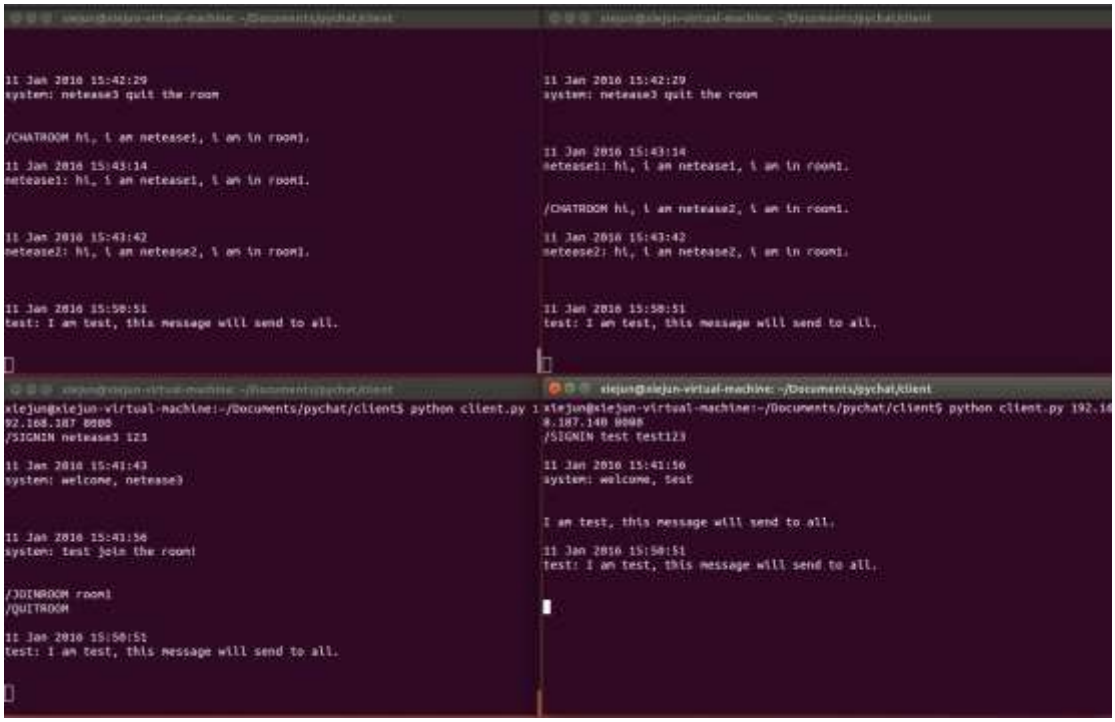
1) 终端 1 中输入 /CREATEROOM room1
2) 终端 2 中输入 /JOINROOM room1
3) 终端 3 中输入 /JOINROOM room1
4) 终端 3 中输入 /QUITROOM
5) 终端 1 中输入 /CHATROOM hi, i am netease1, i am in room1.
6) 终端 2 中输入 /CHATROOM hi, i am netease2, i am in room1.





在房间中，直接不带命令发送的消息也是朝向大厅的，使用账户 test 用户发送 I am test,

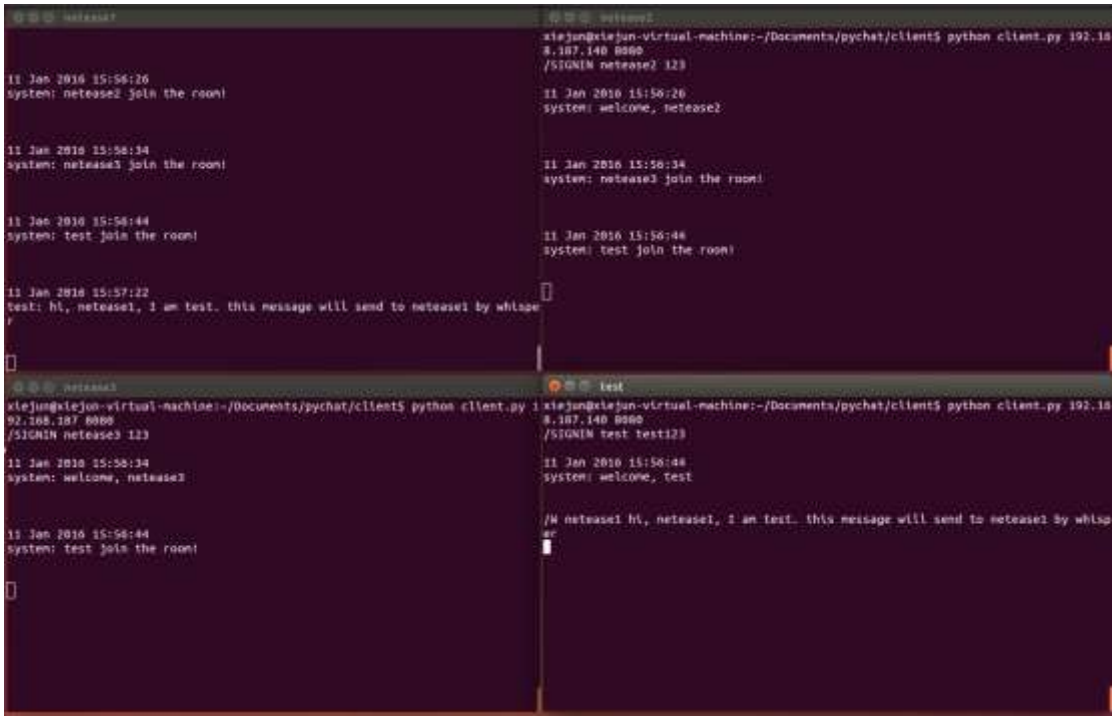
this message will send to all.



7. 私聊

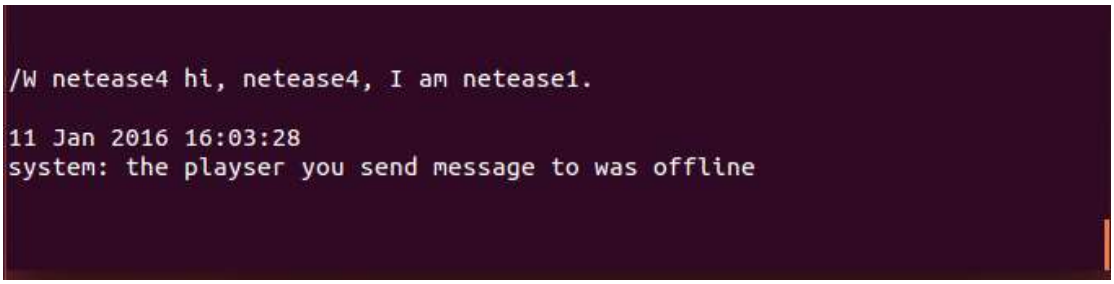
1) 使用 test 用户朝 netease1 用户发送消息，命令如下

/W netease1 hi, netease1, I am test. This message will send to netese1 by whisper.



2) 朝不存在的用户或者不在线的用户发送消息

/W netease4 hi, netease4, I am netease1.

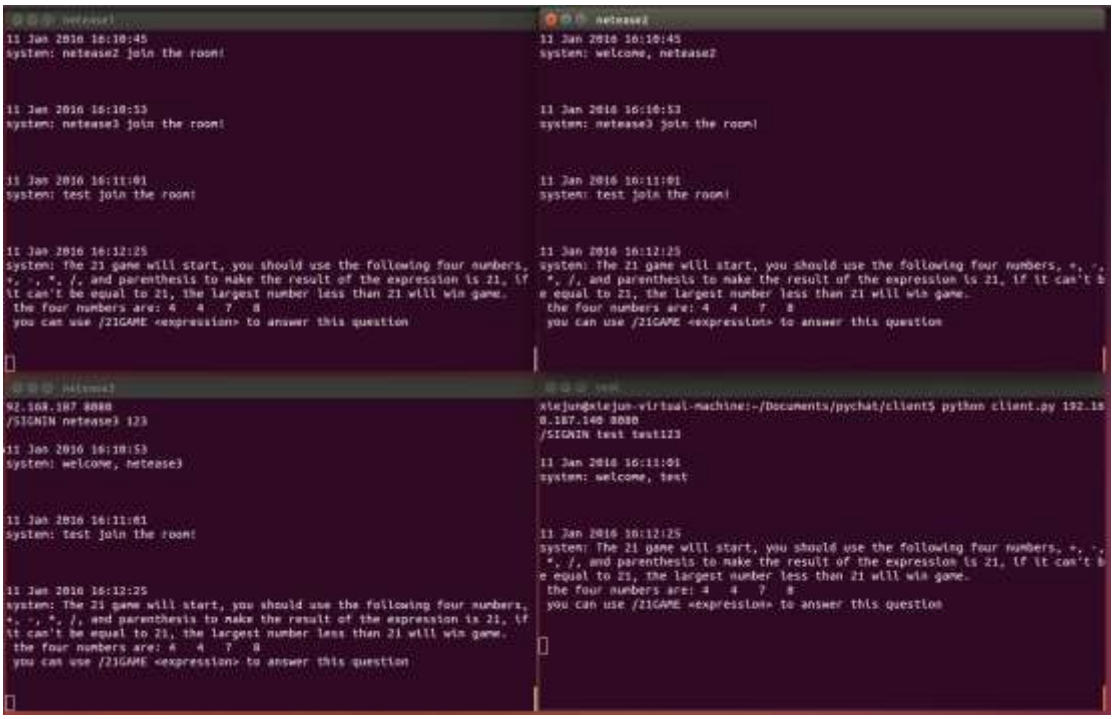


8. 21 点小游戏

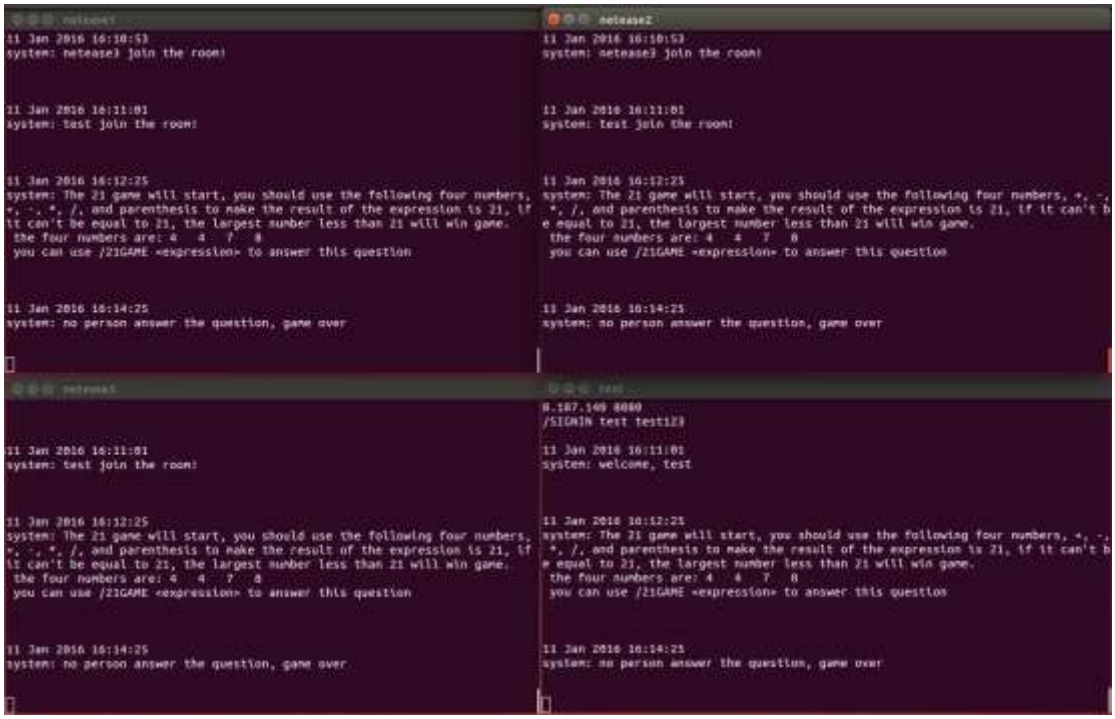
为了方便测试，重启服务端，把游戏开始时间设置为服务器启动后 2 分钟，游戏频率设置为 5 分钟一场，每场游戏持续时间设置为 2 分钟，

```
# the period of the 21 game, second
GAME21_PERIOD = 300

# the duaration time of the 21 game
GAME21_DUARATION = 120

# the first expires of the 21 game, if it set to None, the game will start after in one minute

GAME21_EXPIRES = time.time() + 120
```

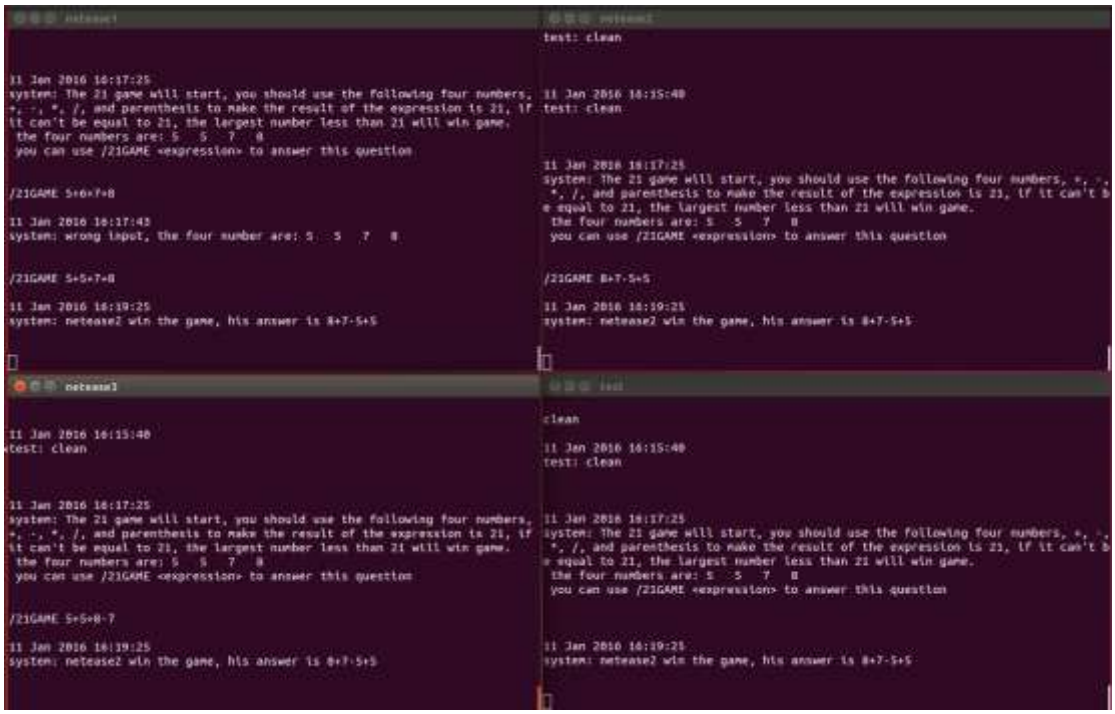
游戏开始



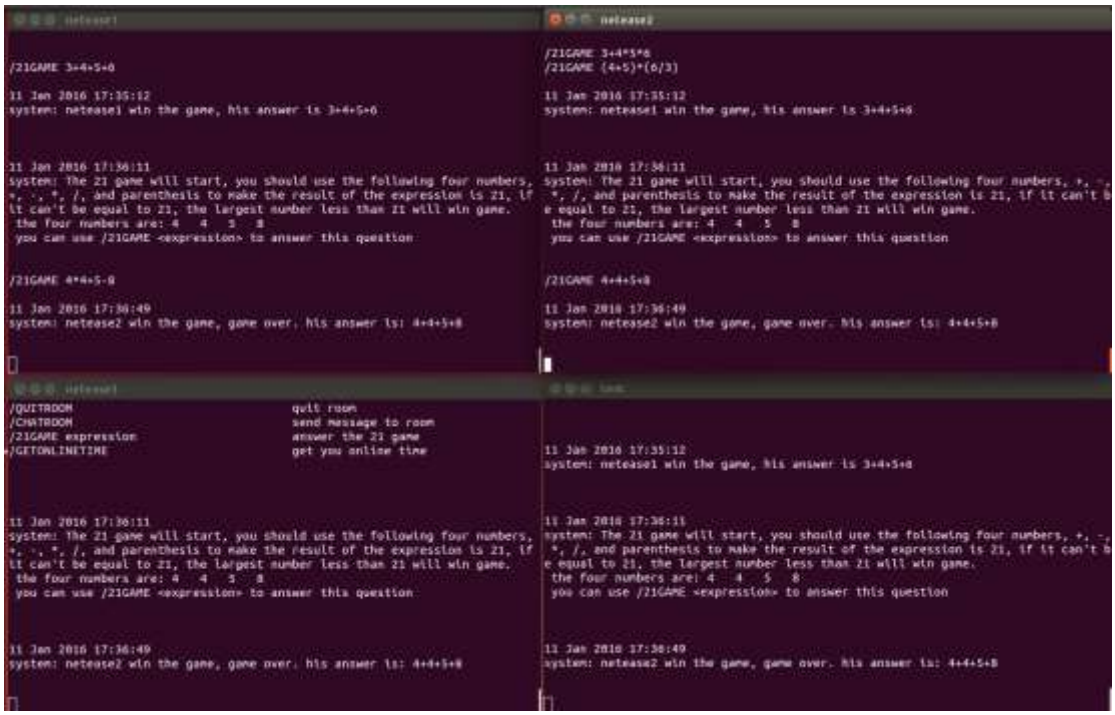
1) 2 分钟到，没人回答



2) 有人回答但没人是 21 点，没人给出正确答案，一个人超过 21， 一个人没答



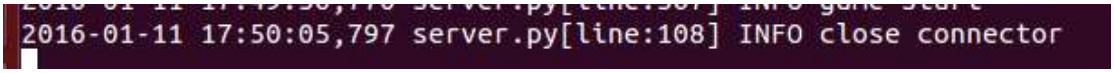
3) 有人在游戏时间内给出等于 21 的答案，游戏立即结束



9. 客户端崩溃，网络连接断开

客户端崩溃服务端感知到后会将用户下线

网络连接异常断开 30 秒后服务器将用户下线，下图即为将测试虚拟机 2 断开网络连接 30 秒后服务端日志，显示将此连接断开



程序实现方案简介

1. 客户端

客户端只做从终端读取输入，从网络读取服务器信息回显，向服务端发送心跳数据几个简单的事。

2. 服务端

服务器负责解析用户发来的消息并处理，监听多个 socket 连接，完成定时任务，处理 21 点游戏逻辑，整个实现使用单线程，用户信息存储在 sqlite 中。

1) 事件循环

网络 IO 编程模型使用 IO 复用，通过 select 来监听多个 socket 连接，有连接就绪就针对此连接的消息进行处理，程序中庸 event_loop 模块的 EventLoop 来实现此功能

2) 客户端服务端消息传递

客户端从终端按行读入用户输入，加入分包的分隔符直接发往服务端，约定/开头的表示为命令，服务端读到分包的分隔符后对消息进行处理，然后发回客户端，客户端收到消息后直接将消息显示到终端上

3) 定时器实现

使用 select 的 timeout 来实现定时器时间基准，由于系统对定时精度要求不高，所以此处直接采用了 1 秒的时间基准。

程序中设计了两种定时器，一种是从某个时间开始，按一定周期触发，21 点游戏即采用这个定时器时间。模块 event_loop 中的类 TaskTimer 为此类定时器的基类；

另一种是设定时间，触发一次后便不会触发，直到再次重置的时候再进行触发，程序里面 21 点游戏的游戏时间结束时采用这个定时器时间的，每次开始游戏后重置这个定时器，定时器到期的时候执行游戏结束操作。模块 event_loop 中的类 TaskTimer2 为此类定时器的基类

4) 应用层心跳

为了处理客户端网络连接断开，服务端无法感知的情况，使用 TCP 的带外数据实现了一个简陋的应用层心跳，客户端定时向服务端发送 1 字节的带外数据，服务端接收到带外数据后将超时标记清零，同时向客户端发送 1 字节带外数据。

服务端使用一个定时器 class HeartBeat(event_loop.TaskTimer) 每 3 秒检查一次所有连接的超时标记并对超时标记+1， 如果检测到超时标记超过 10 则判断此连接出现问题，退出此连接对应的用户，关闭 socket 连接

5) 21 点游戏

21 点游戏处理逻辑通过类 Game21 来处理

3. 可以改进的地方

1) 服务端和客户端消息的交互，当前使用纯文本的方式，客户端收到消息直接回显。可以考虑将消息指定为 json 格式，在消息中包含消息类型，消息目的地，消息内容等信息
2) 定时器实现的时候可以按到期时间为 key，将定时器存储到优先级队列中，这样在定时任务较多的时候不用每次遍历所有定时器查看是否可执行
3) 对用户在线时间的更新现在是当用户退出或者断开的时候才进行更新，可以改为由服务端使用一个定时器定时对所有在线的用户进行在线时间更新
4) 服务端可以考虑使用多个线程实现，比如更新在线时间任务可以在一个单独的线程中完成