

# 一、系统概要

## 1. 系统概述

随着 Internet 的飞速发展，网络安全的地位日益突出。网络的安全措施应是能全方位地针对各种不同的威胁，这样才能确保网络信息的保密性、完整性和可用性。作为安全服务中的一种----实体认证尤为重要。

在一个公开的分布式网络环境中，工作站上的用户希望访问分布在网上的服务器资源。但网络上的资源仅允许授权用户的特定权限的访问，因此，在分布式网络中，必须提供一种机制来对用户的身份进行认证。

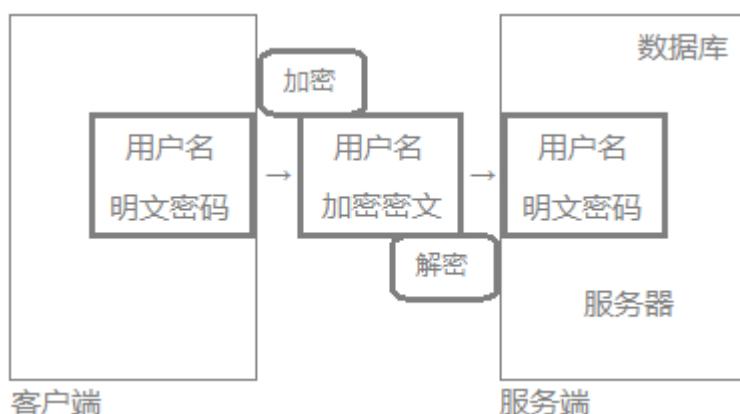
现在常用的一些认证协议是为 TCP/IP 网络设计的基于 Client/Server 模式的三方验证协议，广泛应用于 Internet 服务的访问，网络中的认证协议服务起着可信仲裁者的作用。认证协议基于对称密码体制或非对称密码体制，可提供安全的实体认证。

## 2. 系统架构

在本系统中，用户在客户机上登录，在登录界面上，输入用户名 User 和密码 Password，用户名以正常字符显示，密码以星号显示，为了防止密码在网上传输被窃听者获取，用户输入的密码在客户机上加密  $E_k(\text{Password})$ ，所以在网上传输的是经过加密的用户密码  $E_k(\text{Password})$ 。

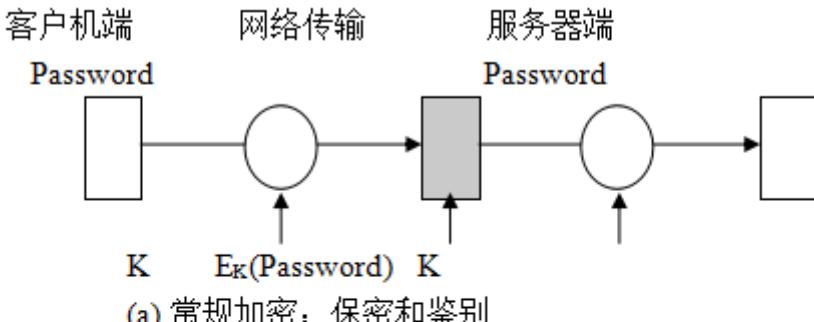
由于偷听者不知密钥  $K$ ，所以即使得到信息  $E_k(\text{Password})$ ，也得不到用户的密码  $\text{Password}$ 。服务器接收客户机传输过来的信息，提取用户名 User 密文  $E_k(\text{Password})$ ，在服务器端对密文解密， $D_k(E_k(\text{Password})) = \text{Password}$ ，得到用户的密码  $\text{Password}$ 。

管理员在服务器数据库中，读出库中存贮的用户名与密码，与接收来的用户名与密码相比较，如果相等，则为合法用户，如果不相等，则为非法用户。

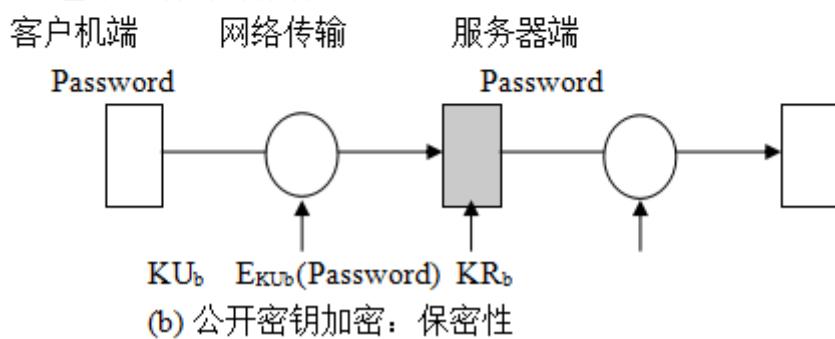


### 3. 加密算法

#### (1) 基于对称密码体制



#### (2) 基于公钥密码体制



### 4. 开发环境

操作系统: 32 位 Ubuntu 15.10 系统, 7.3 GiB

数据库系统: MySQL

程序语言: Python 2.7

开发工具: Boa Constructor

## 二、通信规范的制定

### 1. 客户端

客户端得到的数据格式为‘用户名+明文密码’，在本地算法 DES/RSA 加密后，通过 socket 传送的数据格式为‘用户名+加密密文’。并携带本次加密的算法名发送。

发送前尝试连接服务端，若服务端连接成功，返回‘Client to Server:Connecting!+系统时间’，连接失败则返回‘Client to Server:Disconnected!’。

等待服务端验证登录并等待输出。验证成功，则输出‘用户名+has joined!’，验证失败，则输

出 ‘Warning’ 。

## 2.服务端

服务端进行数据库检索操作，数据库检索无误则返回 ‘select \* from LoginTable’，即 LoginTable 表的各项数据。检索出错，则报错 ‘Mysql Error...’ 。

开始监听，日志输出 ‘Listening’ 。结束监听，日志输出 ‘UnListen’ 。

监听后开始验证客户端发来的数据，解密后进行数据匹配验证，验证成功，则返回 ‘用户名+has joined!’，日志输出 ‘算法名+' "Usr:'+用户名+' Port:(+加密密文+) Paw:'+明文密码+" Logged’。验证失败，则返回 ‘Warning’，日志输出 ‘算法名+' "Usr:'+用户名+' Port:(+加密密文+) Paw:'+明文密码+" Error!!’ 。

## 三、算法设计和实现

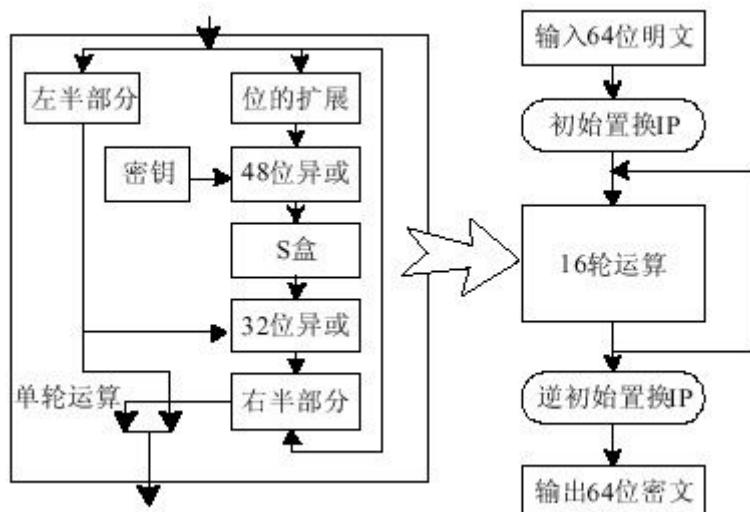
### 1.DES 算法

(1)算法设计：

假定明文 m 和密钥 k 都是 64 比特的 0, 1 符号串。

设  $m=m_1m_2\cdots\cdots m_{64}$ ,  $k=k_1k_2\cdots\cdots k_{64}$  ( $m_i, k_i=0$  或  $1, i=1, 2, \dots, 64$ ) 。

加密过程表达如下： $DES(m)=IP-1 \cdot T_{16} \cdot T_{15} \cdot \cdots \cdot T_2 \cdot T_1 \cdot IP(m)$  迭代循环次数为 16 轮，其中 T 是每轮迭代。



(2)算法实现：

定义函数，依次实现 16 轮迭代，

```
def __code(self, s, k):
    s = self.__IP(s)
    l, r = s[0:32], s[32:64]
    for i in range(16):
        r_t = r
        r = self.__E(r)
        r = self.__F(r, k[i])
        r = self.__S(r)
        r = self.__P(r)
        r = self.__F(r, l)
        l = r_t
    return self.__class__.__IP1(r+l)
```

并实现加密、解密函数。

## 2.RSA 算法

(1)算法设计：

使用 RSA 有三个阶段。

RSA 阶段 1：确定公钥和私钥

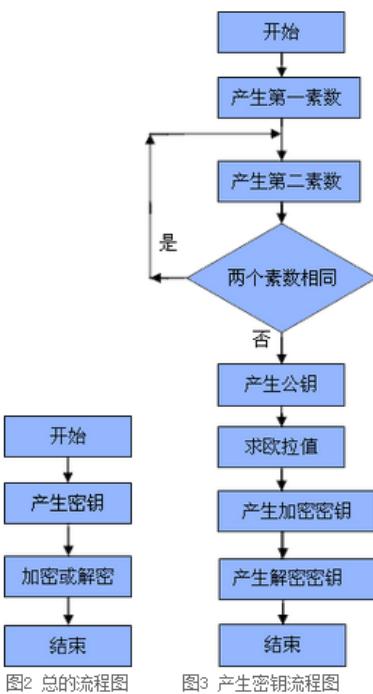
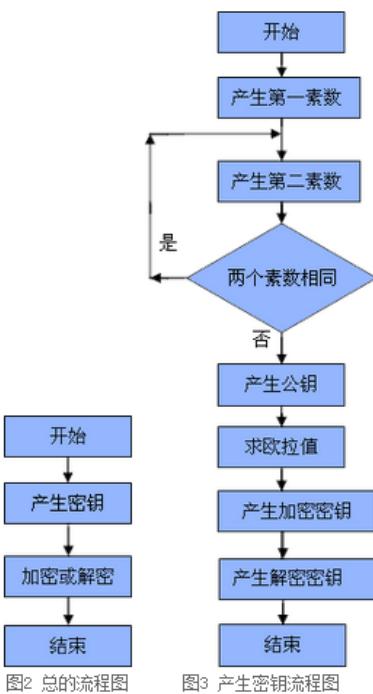
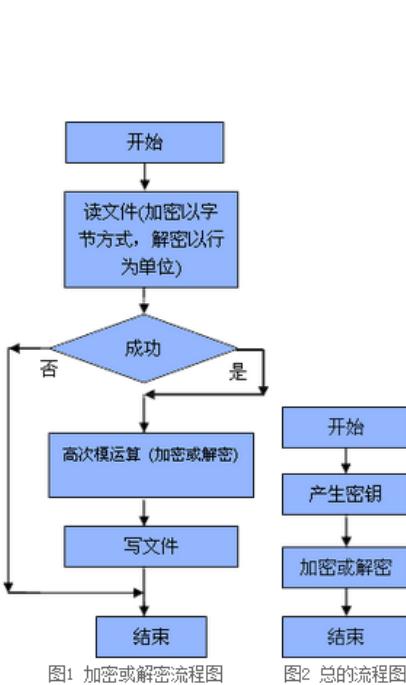
- ✓ 选择两个大素数， P 和 Q。
- ✓ 计算  $N = P * Q$ 。
- ✓ 计算  $f(n) = (P - 1)(Q - 1)$ 。
- ✓ 选择  $e$ ， 其中  $1 < e < n-1$  且  $\text{GCD}(e, f(n)) = 1$ 。
- ✓ 计算  $d$ ， 其中  $ed \equiv 1 \pmod{f(n)}$  （使用扩展的欧几里德算法）。
- ✓  $(e, n)$ 作为公钥，  $(d, n)$ 作为私钥。

RSA 阶段 2：加密信息

使用 RSA 加密消息 M，进行下列加密运算：  $C = M \text{mod } n$ ,其中 C 是密文，发送 C。

RSA 阶段 3：解密信息

使用 RSA 解密密文 C，进行下列解密运算为：  $M = C \text{mod } n$ ,其中 M 是原始明文。



## (2)算法实现:

```

class PublicKey():
    __slots__ = ('n', 'e')
    def __init__(self, n, e):
        self.n = n
        self.e = e
    def __getitem__(self, key):
        return getattr(self, key)
    def __repr__(self):
        return 'PublicKey(%i, %i)' % (self.n,
                                       self.e)
    def __eq__(self, other):
        if other is None:
            return False
        if not isinstance(other, PublicKey):
            return False
        return self.n == other.n and self.e
            == other.e

def __ne__(self, other):
    return not (self == other)

class PrivateKey():
    __slots__ = ('n', 'e', 'd')
    def __init__(self, n, e, d):
        self.n = n
        self.e = e
        self.d = d
    def __getitem__(self, key):
        return getattr(self, key)
    def __repr__(self):
        return 'PrivateKey(%(n)i, %(e)i, %(d)i)' % self
    def __eq__(self, other):
        if other is None:
            return False
        if not isinstance(other, PrivateKey):
            return False

```

```
return False                                self.d == other.d)

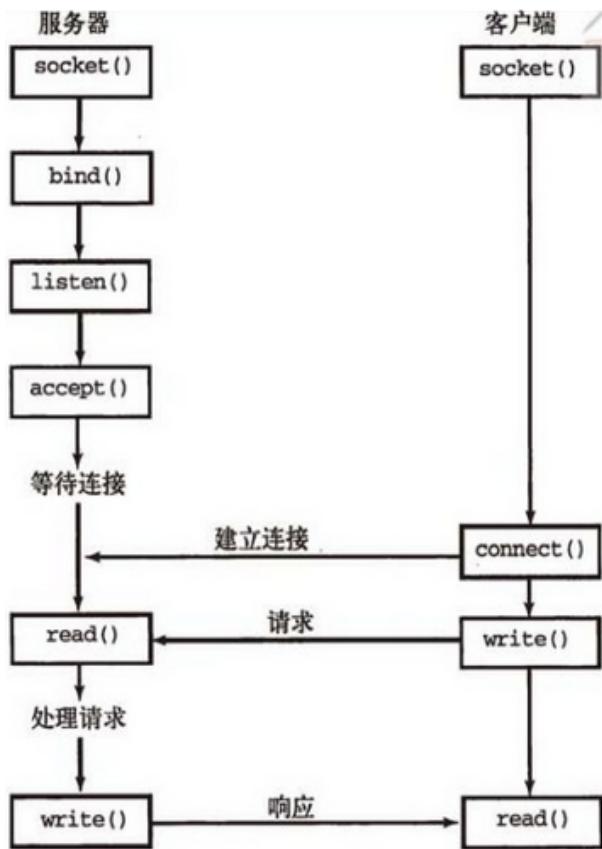
return (self.n == other.n and                def __ne__(self, other):
        self.e == other.e and                  return not (self == other)
```

实现 $(e, n)$ 作为公钥， $(d, n)$ 作为私钥。

### 3.核心算法：socket 实现通信

(1) 算法设计:

Python 提供了基本 Socket 模块，它提供了标准的 BSD Sockets API。



服务端：

- 创建套接字，绑定套接字到本地 IP 与端口

```
# socket.socket(socket.AF_INET,socket.SOCK_STREAM), s.bind()
```

- ### ● 开始监听连接

```
#s.listen()
```

- 进入循环，不断接受客户端的连接请求

#s.accept()

- 然后接收传来的数据，并发送给对方数据

```
#s.recv(), s.sendall()
```

- 传输完毕后，关闭套接字

```
#s.close()
```

客户端：

- 创建套接字，连接远端地址

```
# socket.socket(socket.AF_INET,socket.SOCK_STREAM), s.connect()
```

- 连接后发送数据和接收数据

```
# s.sendall(), s.recv()
```

- 传输完毕后，关闭套接字

```
#s.close()
```

(2)算法实现：

#client.py 客户端：

```
if __name__ == '__main__':
    import socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('localhost', 8001))
    sock.send('1')
    print sock.recv(1024)
    sock.close()
```

#### 4.其他

服务端线程和监听的实现。因为预期设计希望打开 Server 服务端就进入 socket 连接，又同时兼有图形界面。所以本程序采用并行线程。图形界面打开主线程进行的同时，socket 服务端子线程也启动。并添加 timer 实现对客户端的监听。

算法实现：

```
#thread t1
```

```
def serverf():
```

```
factory = Factory()

#socketT = IphoneChat()

#socketT.sbutton(self.richTextCtrl1)

#factory.protocol = module1.SocketT

factory.protocol = With

factory.clients = []

reactor.listenTCP(8001, factory)

reactor.run()

t1 = threading.Thread(target=serverf,args=())

threads.append(t1)

#main

def __init__(self, parent):

    self.__init__ctrls(parent)

    self.Inputs('Welcome')

    #running

    for t in threads:

        t.setDaemon(True)

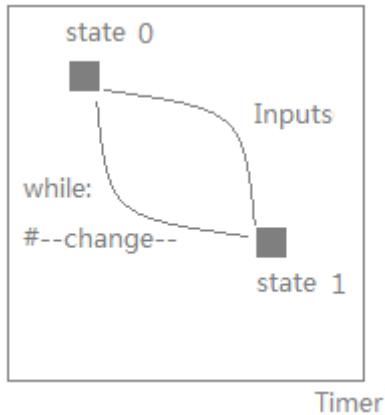
        t.start()

    self.Inputs('Server started')
```

## 四、主要模块的设计分析

### 1.服务端监听模块

使用状态位 state 监听，获取后就将 state 改为 1.而监听程序一旦监听到 state 为 1，执行输出程序，即后就将 state 改为 0.



```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <vector>
#include <Windows.h>
#include <mysql.h>
using namespace std;

class Infomsg {
public:
    string ing;
    int state;
};

Infomsg infomsg;

void OnInput(Infomsg& msg) {
    cout << "Input received: " << msg.ing << endl;
}

void OnTimer1Timer(Infomsg& msg) {
    if(msg.state == 0) {
        cout << "State 0" << endl;
        msg.state = 1;
    } else {
        cout << "State 1" << endl;
        msg.state = 0;
    }
}

int main() {
    Infomsg msg;
    msg.state = 0;
    msg.ing = "Initial Input";
    OnInput(msg);
    OnTimer1Timer(msg);
}

```

## 2. 服务端数据库模块

服务端连接数据库并实现对数据库中用户名和密码的检索并查看。

```

def OnButtonselectButton(self, event):
    try:
        conn=MySQLdb.connect( host='localhost' ,user='root', passwd='xuyifan', db='infologin',
        port=3306)
        cur=conn.cursor()
        strout=cur.execute('select * from LoginTable')
        #print strout
    except MySQLdb.Error, e:
        print "Mysql Error %d: %s" % (e.args[0], e.args[1])

```

```

        self.Inputs('There are '+str(strout)+' Users')

        info=cur.fetchmany(strout)

        for i in info:

            #print i

            self.Inputs(str(i))

        cur.close()

        conn.close()

    except MySQLdb.Error,e:

        self.Inputs('Mysql Error '+str(e.args[0])+':'+e.args[1])

#____Sql_____

event.Skip()

```

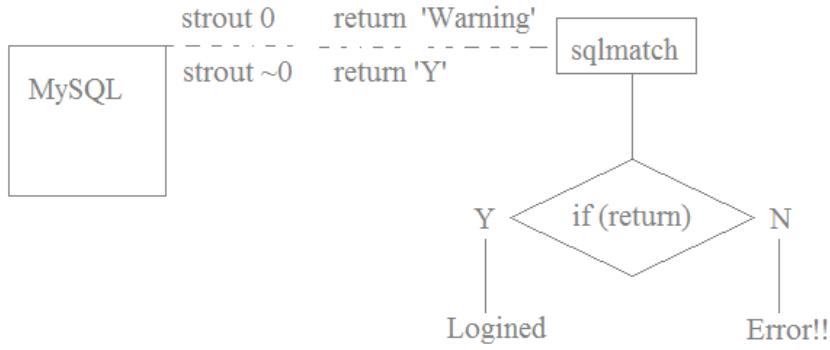
### 3.服务端日志模块

采用 richTextCtrl 控件对服务端信息进行输出操作，实现服务端日志模块。服务端更新信息时，执行 self.richTextCtrl1.SetValue(self.richTextCtrl1.GetValue()+'\n'+times+' : '+string)，更新其中的内容并实现添加当前时间功能。



### 4.服务端匹配模块

使用数据库 select 语句实现对获取到的用户名和密码进行有效性匹配。在服务端根据每次匹配结果判断该用户是否为合法用户。



```

#.....
#infomsg.sqlmatch

        strout=cur.execute('select * from LoginTable Where User ='+"'"+checkusr+"'"+'and Password ='+"'"+checkpaw+"'")

        if(strout!=0):
            return 'Y'

        else:
            return 'Warning'

#.....
#.....
#server.py

        sqlkey=infomsg.sqlmatch(command,paw)

        if(sqlkey=='Y'):

            msg=command+' has joined!'

            pri=count+' "Usr:' + command + ' Port:(+' + content + ')' + Paw:' + paw + '" Logined'

        else:

            msg=sqlkey

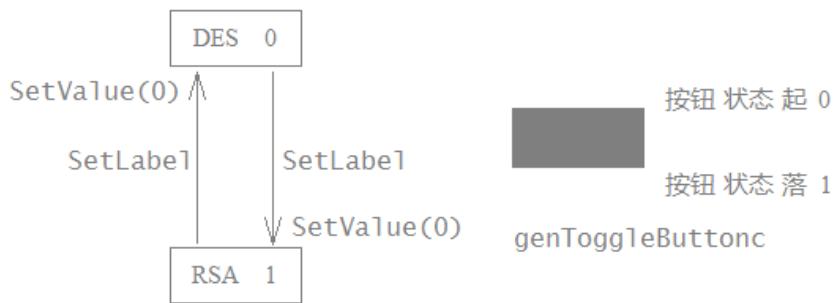
            pri=count+' "Usr:' + command + ' Port:(+' + content + ')' + Paw:' + paw + '" Error!!'

#.....

```

## 5.客户端算法选择模块

通过浮动按钮 `self.genToggleButtonc` 实现，默认显示为‘DES’。当按钮被按下后，通过其定义值将其修改为状态 0，即‘RSA’选择模式。当按钮再次被按下后，同理，切换为‘DES’选择模式。



```

def OnGenToggleButtoncButton(self, event):
    if(1):
        if(self.genToggleButtonc.GetLabel()=='DES'):
            self.genToggleButtonc.SetLabel('RSA')
            self.genToggleButtonc.SetValue(0)
        elif(self.genToggleButtonc.GetLabel()=='RSA'):
            self.genToggleButtonc.SetLabel('DES')
            self.genToggleButtonc.SetValue(0)
        #turn change
        event.Skip()
    
```

## 6.客户端登录模块

使用登录模块登录时，先根据选择的算法以及手动填写的用户名和密码，密码进行加密，然后将算法+用户名+加密密文通过”：”连接后 socket 传送。然后服务端可以通过字符串切片将字符串中各个部分解出。



如 DES 算法的 `sock.send('DES:' + self.textCtrluser.GetValue() + ':' + paw)`

```

def OnButtonloginButton(self, event):
    times=time.strftime("%H:%M:%S", time.localtime())
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect(('localhost', 8001))
        self.printtext.SetLabel('Client to Server:Connecting!' + times)
        #send
        if(self.genToggleButtonc.GetLabel()=='DES'):
            paw=self.textCtrlpassword.GetValue()
            #DES
            d=des.DES()
            d.input_key(des.key)#8
            paw=d.encode(paw)
            #DES
            sock.send('DES:' + self.textCtrluser.GetValue() + ':' + paw)
        elif(self.genToggleButtonc.GetLabel()=='RSA'):
            paw=self.textCtrlpassword.GetValue()
            #paws=str(rsa.n+rsa.e)

```

```

#paws=rsa.a()

pawning=str(paw)

paws=rsa.encrypt(pawning,rsa.n,rsa.e)

sock.send('RSA:' +self.textCtrluser.GetValue() +':'+paws)

#recv

self.staticTextrev.SetLabel(sock.recv(1024))

#print sock.recv(1024)

sock.close()

except:

    self.printftext.SetLabel('Client to Server:Disconnected!')

    #print 'Error'

    #error input

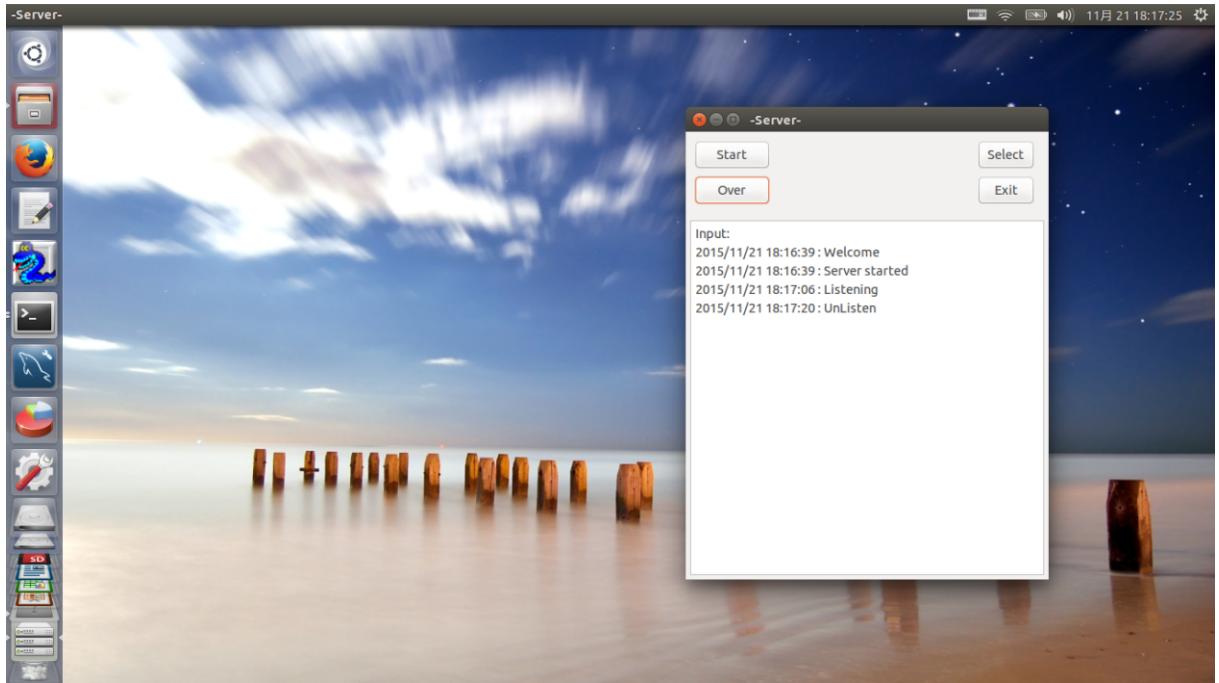
    event.Skip()

```

## 五、系统运行效果

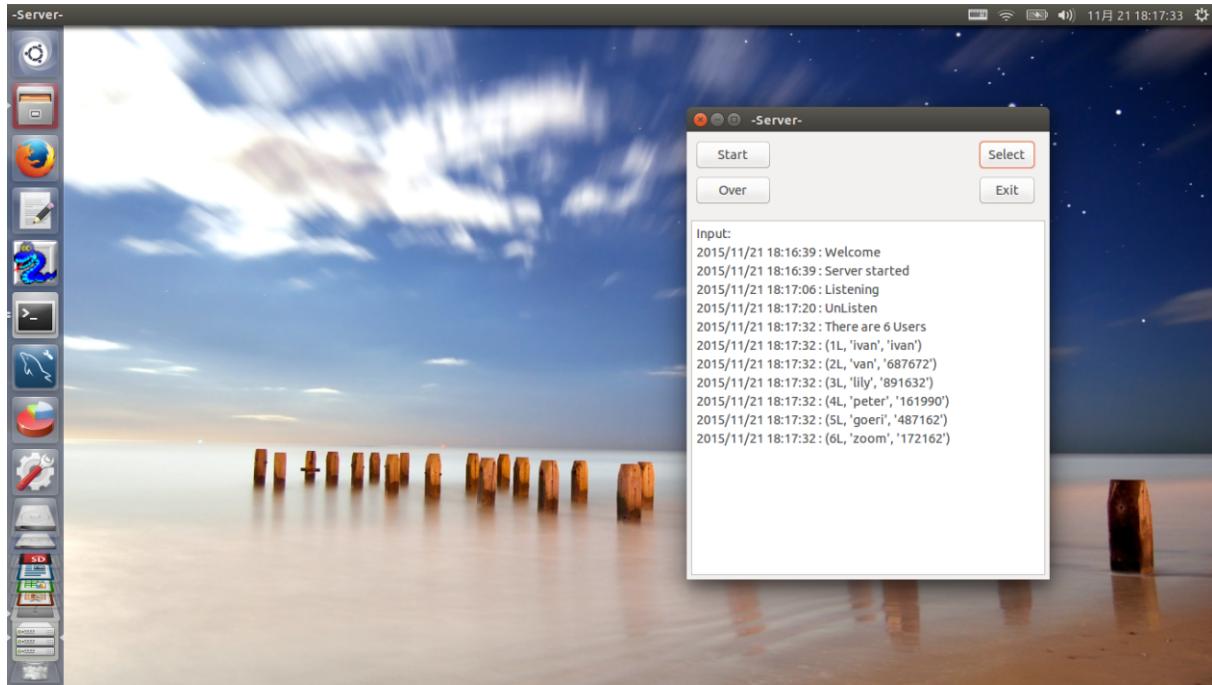
### 1.服务端监听信息

输出：2015/11/21 18:17:06 : Listening, 2015/11/21 18:17:20 : UnListen



## 2.服务端检索数据库

检索数据库，输出各个合法用户的信息（用户名，密码）

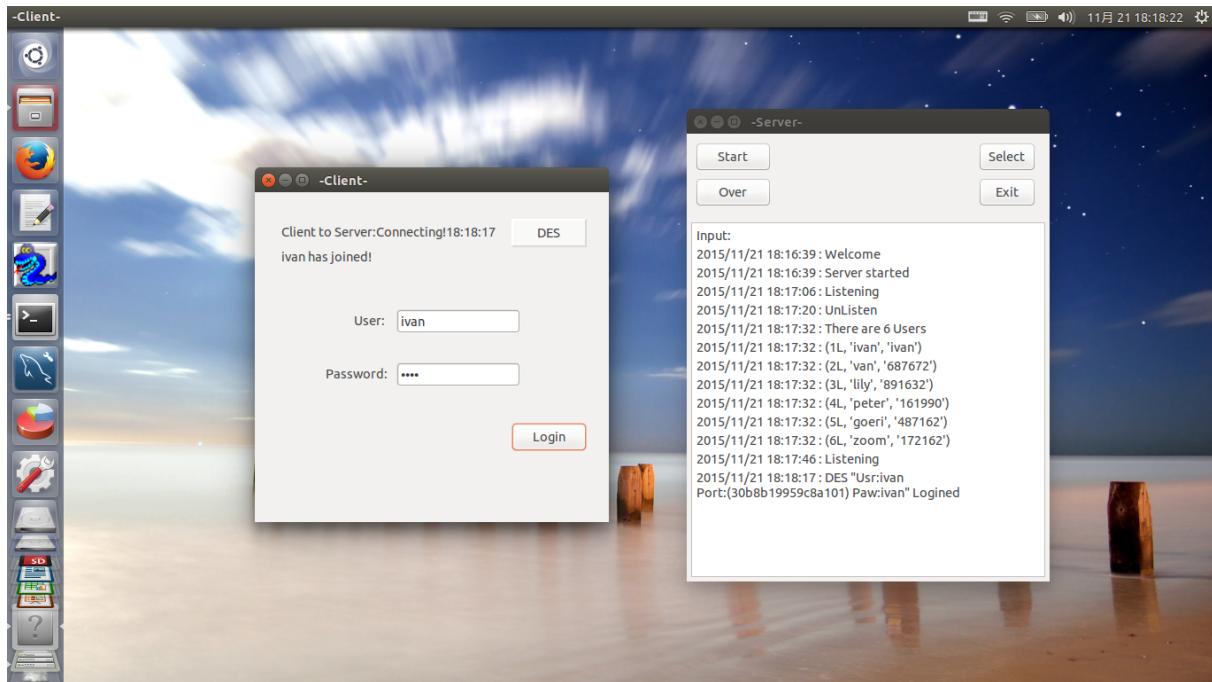


## 3.客户端登录

(1) DES，合法用户登录

合法用户 ivan（密码 ivan）登录，输出：

2015/11/21 18:18:17 : DES "Usr:ivan Port:(30b8b19959c8a101) Paw:ivan" Logined

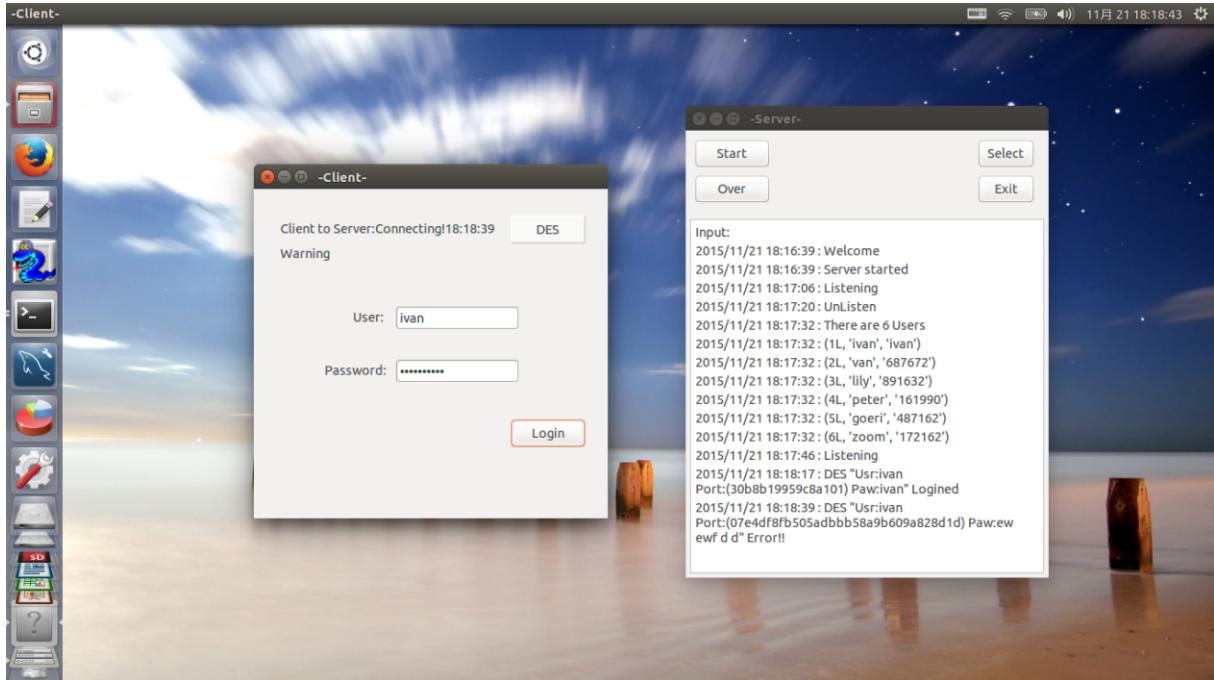


## (2) DES, 非合法用户登录

非合法用户 ivan (错误密码 ew ewf d d) 登录, 输出:

2015/11/21 18:18:39 : DES "Usr:ivan Port:(07e4df8fb505adb58a9b609a828d1d) Paw:ew ewf d d"

Error!!

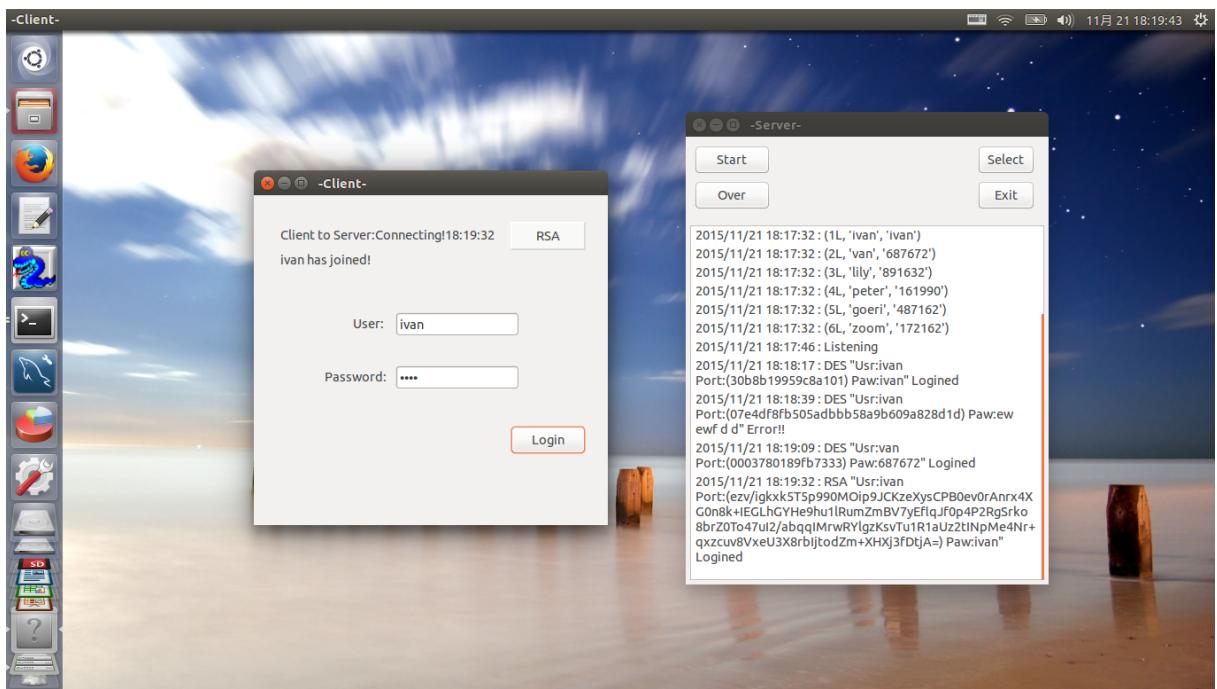


## (3) RSA, 合法用户登录

合法用户 ivan (密码 ivan) 登录, 输出:

2015/11/21 18:19:32 : RSA "

Usr:ivanPort:(ezv/igkxk5T5p990MOip9JCKzeXysCPB0ev0rAnrx4XG0n8k+IEGLhGYHe9hu1lRum  
ZmBV7yEflqJf0p4P2RgSrko8brZ0To47uI2/abqqIMrwRYlgzKsvTu1R1aUz2tINpMe4Nr+qxzcuV8VxeU3  
X8rbIjtodZm+XHXj3fDtjA=) Paw:ivan" Logined



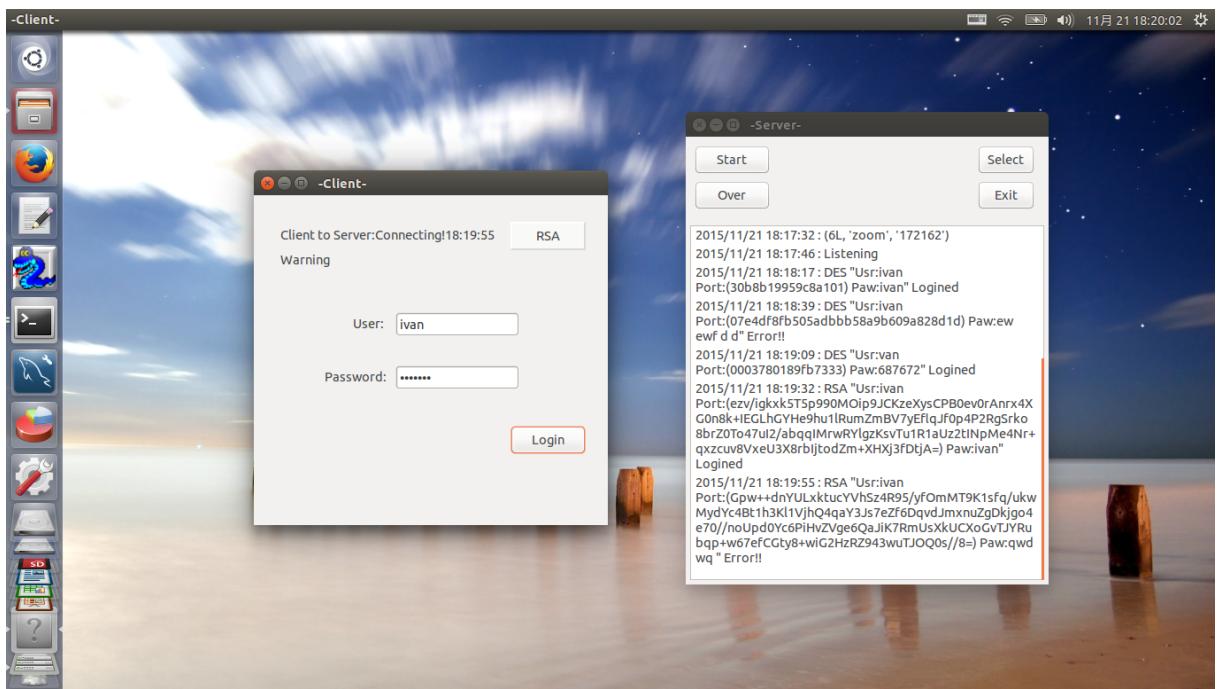
#### (4) RSA, 非合法用户登录

合法用户 ivan (密码 qwd wq) 登录，输出：

2015/11/21 18:19:55 : RSA "

Usr:ivan

Port:(Gpw++dnYULxktucYVhSz4R95/yfOmMT9K1sfq/ukwMydYc4Bt1h3K11VjhQ4qaY3Js7eZf6DqvJ  
mxnuZgDkjgo4e70//noUpd0Yc6PiHvZVge6QaJiK7RmUsXkUCXoGvTJYRubqp+w67efCGty8+wiG2Hz  
RZ943wuTJOQ0s//8=) Paw:qwd wq " Error!!



#### 4. 多客户端登录

启动三个客户端，分别使用用户 ivan（密码 ivan）DES 加密登录，用户 van（密码 687672）DES 加密登录，用户 zoom（密码 172162）RSA 加密登录。可以在服务端正常输出查看登录信息。

输出：

2015/11/21 18:26:10 : Listening

2015/11/21 18:26:10 : DES "Usr:ivan Port:(30b8b19959c8a101) Paw:ivan" Logged

2015/11/21 18:26:18 : DES "Usr:van Port:(0003780189fb7333) Paw:687672" Logged

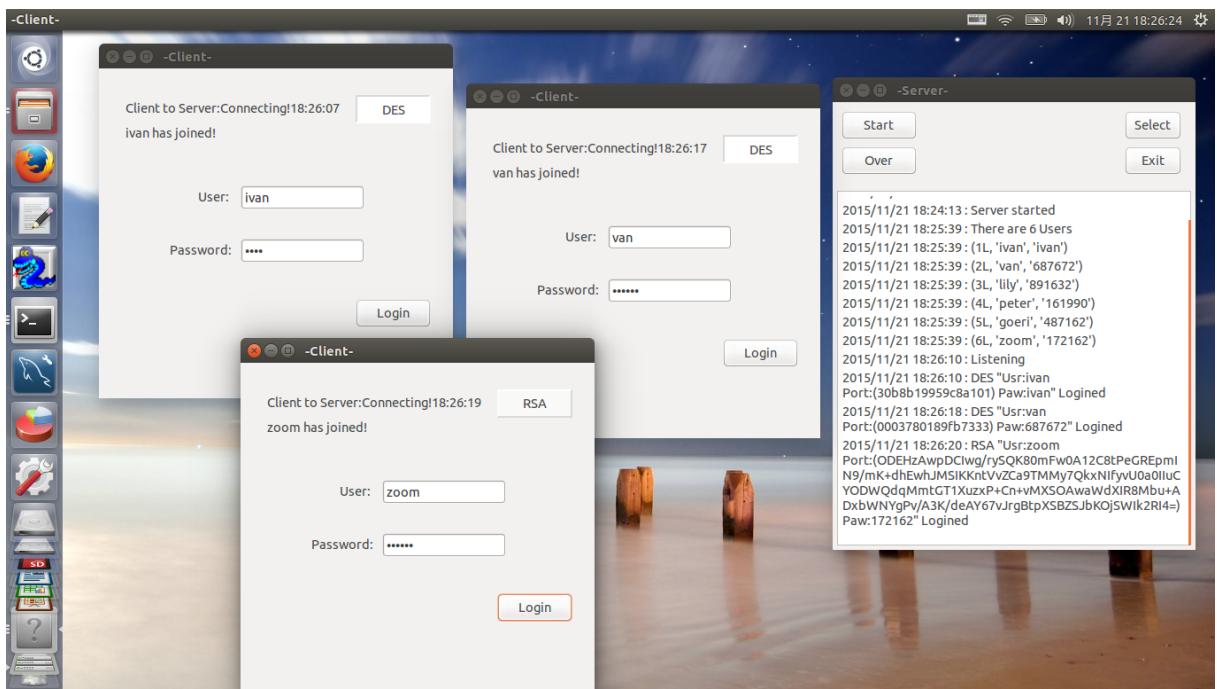
2015/11/21 18:26:20 : RSA "

Usr:zoom

Port:(ODEHzAwpDCIwg/rySQK80mFw0A12C8tPeGREpmIN9/mK+dhEwhJMSIKKntVvZCa9TM

My7QkxNIfyvU0a0IIuCYODWQdqMmtGT1XuzxP+Cn+vMXSOAwaWdXIR8Mbu+ADxbWNYgP

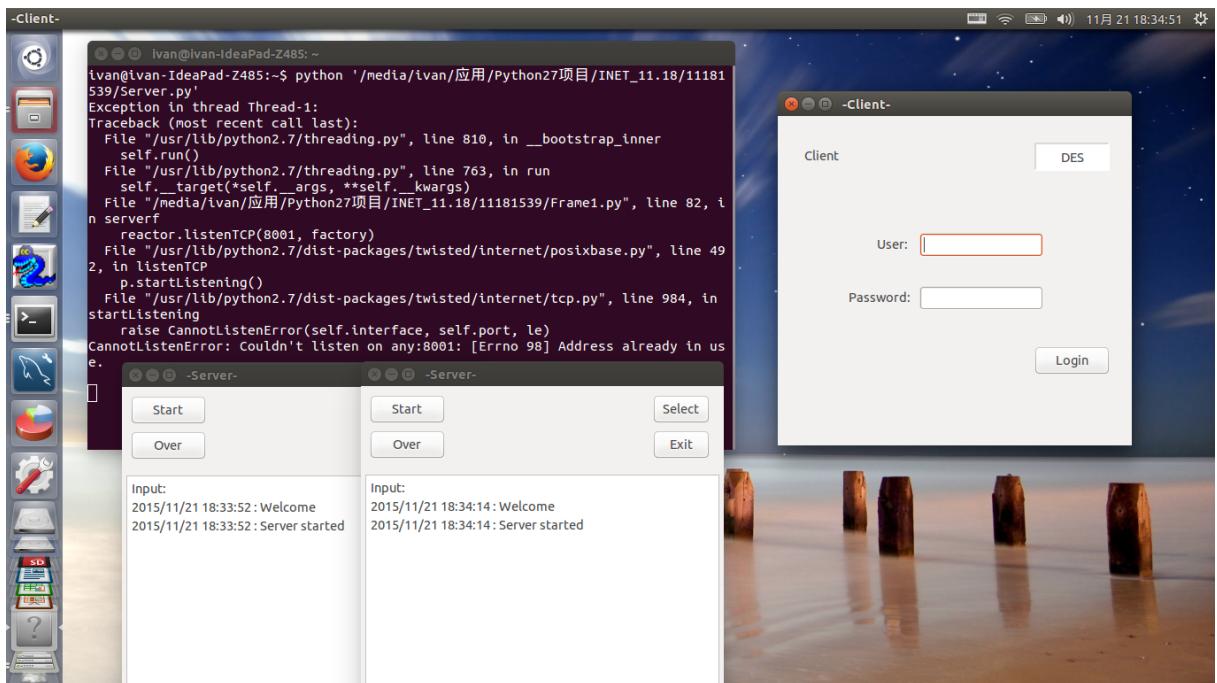
v/A3K/deAY67vJrgBtpXSbzSJbKOjSWIk2RI4=" Paw:172162" Logged



## 5.多服务端报错

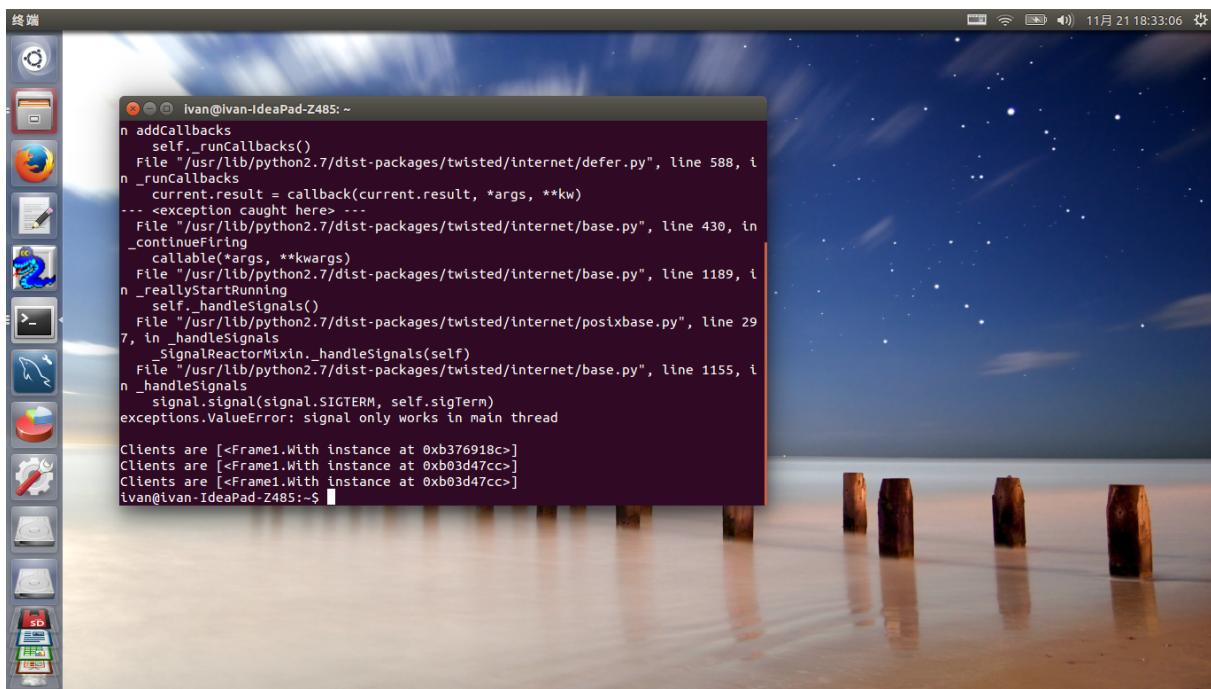
多服务端建立，由于端口地址 8001 被前一个服务端占用，所以报错如下：

```
CannotListenError: Couldn't listen on any:8001: [Errno 98] Address already in use.
```



## 6.其他

通过在客户端建立于服务端连接时，添加代码'Clients are',self.factory.clients，然后对上述建立三个客户端时，代码中添加代码检测三个客户端的使用情况。内存中使用情况输出如下：



## 六、心得与体会

### 1. 项目存在的问题

#### (1) 监听机制

程序中采用主线程和子线程的方式，将 server 的启动放在子线程之中，每次主线程即图形界面启动时，子线程就同时被启动了。这也就导致如果光是打开了服务端而一直没有客户端与服务端建立连接，调试程序会结束之后提示子线程没有启动。但这个报错并不是程序错误，不会影响到程序的启动和有客户端使用情况下的调试。可以通过设置 connection.settimeout(x) 来实现对 socket 消息的时限回复 timeout。本程序以模拟为主，不完全模拟服务器操作，所以选择忽略此问题。

#### (2) 模块完善不足

程序客户端模块还不够完善，应该进一步添加响应功能。例如用户名或密码为空则提前报错，或者先返回用户名即直接进行数据库搜索，若用户不存在则直接报错而不进行加密传输。这样可以在程序响应的角度上有进一步优化。

### 2. 实现过程遇到的问题

#### (1) 端口问题

最开始编写程序时，没有实现在每次程序跑完之后就 kill 该程序所在的线程。这导致程序的每

一次调试就会占用到一次端口 8001，而且该端口在程序调试结束之后仍然被线程占用。影响后一次调试和运行，会出现上述多服务端运行中出现的，地址被占用而程序不用正常运行的问题。

### (2) 线程问题

本程序采用的监听模块在程序一开始就通过主线程和子线程的机制，使得服务端模块在图形界面一开始就被启动。这是在多次调试程序后采用的方法，最开始没有考虑到并发线程，所以服务端一开始，就导致进程一直在服务端处跑，客户端显示连接成功，而图形界面没有启动。改进后，线程、监听和图形界面并发进行。

### (3) 类的嵌套

本程序使用 Python 语言实现，在程序界面主类中本欲使用嵌套类的方法来实现监听的子类，因为这样可以更好地调用程序界面中的 richtext 控件来输出设计好的日志模块。随后发现，在本程序开发环境下，嵌套类的方法被限制使用。多次调试排错后放弃嵌套类的实现方法，通过添加全局变量的方式改良实现。

## 3. 相关思考

(1) 服务器数据库存储的是用户的口令，如果攻击者攻击了服务器的数据库系统，获取了用户的口令 Password，则攻击此认证协议成功，有无改进办法？

改进方法：不在数据库直接存入用户的明文口令，而是本地有加密机制可以实现对密码的加密操作，采用如 MD5 或其他加密算法。这样即使攻击者攻击了服务器的数据库，用户口令也不会泄漏。另外数据库系统本身的安全机制也是需要考虑的，这能从根本上解决这个问题。

### (2) 如何进一步完善该认证系统？

针对上述模块功能不足的部分，可以完善、修改登录模块的响应优化。再则监听模块已经实现开始监听和暂停监听功能，但仅限于模拟监听实现。还可以通过 socket 命令的方式修改实现，这样可以进一步优化。再则就是算法实现部分，这部分由于本程序的返回比较简单，仅是用户登录提醒。所以没有很多代码处理，所以如果是比较复杂的算法实现机制，建议在响应上分块实现，这不仅有利于代码可读性，也有利于本认证系统的整体安全性。