Note about the connection and client – server session :

Overview:

When server installed to a machine, I want it listening on all network adapter.
So the IP of server socket would be bind with '0.0.0.0'
List of designed request message from client to server:

```
LOGIN <username> <password>

REGISTER <username> <password>

MSG <src_username> <des_username> <content> (*)

GETLIST
```

(*) Command for client's user is "`MSG <des_username> <content>`" Then client convert it to `MSG <src_username> <des_username> <content>` and send to server.

In this note, we're going to go through the backbone of server and client. I will not show unnecessary infomation like "string processing"

Server:

Server must have an authentication module, socket:

```
import socket
import Authenticate

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
...
auth_module = Authenticate.Authenticate()
...
```

Generally, server handle the connection by monitoring all socket in infinite loop, we can monitor it by **select()** method:

```
while True:
        # Monitor all sockets and detect activity on that sockets
        readable_socks, writable_socks, err = select.select(
                                              [<socket of server + sockets of client>],
                                              [<sockets of client>],
                                              [<sockets may happen error>])
        # Process all sockets which is readable
        # I call this "Readable Loop" in this document
        for sock in readable_socks:
                ...
        # Process all sockets which is wriable
        # I call this "Writable Loop" in this document
        for sock in writable_socks:
                ...
        # Process all sockets which is happen error
        # I don't cover this case in this project
        for sock in writable_socks: # i don't cover this case in this project
                ...
```

First, monitor the sockets may happen event by an incoming message arrived, so that sockets would be server socket and client socket (obviously, client socket use for bidirectional connection). Then we have:

```
watches_incoming = [server_sockets]
...
while True:
        readable_socks, writable_socks, err = select.select(watches_incoming, [] , [])

        ...
        socket_conn, tuple_client_addr = server_socket.accept()
        watches_incoming.append(socket_conn)
        ...
```

Secondly, monitor the sockets may happen event when it able to send message, so that sockets would be client sockets. Then we have :

```
clients = []
watches_incoming = [server_sockets]
...
while True:

        readable_socks, writable_socks, err = select.select(watches_incoming, clients ,[])

        ...
        socket_conn, tuple_client_addr = server_socket.accept()
        watches_incoming.append(socket_conn)
        clients.append(socket_conn)
        ...
```

Look back to server socket, when we call accept() for new connection and return the client socket "socket_conn"?

➔ The server socket will be monitored by select() method, when a clients connect to this socket or send a message to this socket, it become readable, then we have:

```
clients = []
watches_incoming = [server_sockets]
...
while True:

        readable_socks, writable_socks, err = select.select(watches_incoming, clients ,[])
        for sock in readable_socks:
                if sock is server_socket:
                        socket_conn, tuple_client_addr = server_socket.accept()
                        watches_incoming.append(socket_conn)
                        clients.append(socket_conn)
                        ...
```

We already handle the case when a connection come in, But how we handle when the message come in? We should add an “else” statement as below:

```
        readable_socks, writable_socks, err = select.select(watches_incoming, clients ,[])
        for sock in readable_socks:
                if sock is server_socket:
                        socket_conn, tuple_client_addr = server_socket.accept()
                        watches_incoming.append(socket_conn)
                        clients.append(socket_conn)
                        ...
                else:
                        # Process the case when an message come in.
                        ...
```

Here is details about message processing, the main idea is when a message come in, process it in “Readable Loop” then put the respone message to a queue, and when the “Writable Loop” start, it get the respone from that queue and send it to client, each client socket has it own message queue:

```
message_queues = {}  # Pair : <client socket> : <queue>
while True:
    readable_socks, writable_socks, err = select.select(watches_incoming, clients ,[])
    for sock in readable_socks:
        if sock is server_socket:
                        socket_conn, tuple_client_addr = server_socket.accept()
                        watches_incoming.append(socket_conn)
                        clients.append(socket_conn)
                        ...
        else:

             data = ''
             try:
                 data = sock.recv(1024)
             except:
                 pass # client crashed, do nothing here and let code below close the connection.
             if data:
                 # A readable client socket has data
                 print 'received "%s" from %s' % (data, sock.getpeername())
                 elements = data.split(" ", 3)
                 prefix = elements[0]
                 if prefix == "LOGIN":  # message template "LOGIN <username> <password>"
                     ...

                 elif # <another message>
                     ...

                 else:
                     message_queues[sock].put("SERVER RESPONSE: Don't know the command")

             else: # data == '', that means the connection is disconnected
                   # close client socket
                 # Stop listening for input on the connection
                 clients.remove(sock)
                 watches_incoming.remove(sock)
                 # To stop examine this sock in ' Writable Loop '(strong logic)
                 writable_socks.remove(sock)
                 sock.close()
                 # Remove message queue
                 del message_queues[sock]
```

After process and put the respone to message queue, the “Writable Loop” just has one mission:
Send it to client, sleep 10ms after a message to avoid coherence between two continous messages :

```
for sock in writable_socks:
    try:
        msg = message_queues[sock].get_nowait()
    except Queue.Empty:
        pass
    else:
        sock.send(msg)
        time.sleep(0.01)  # sleep 10ms
```

Client:

- Client has two threads, main thread for input from keyboard, second thread for handling incoming message.
- Client just read from keyboard and send to server, then receive the respone, so user must enter defined command below to work with server:

```
LOGIN <username> <password>

REGISTER <username> <password>

MSG <des_username> <content>

GETLIST
```

Second thread designed as below,it’s need to wait for message coming, so i use **sock.recvfrom()**

```
class IncomingThread(threading.Thread):

    def __init__(self, sock):
        threading.Thread.__init__(self)
        self.sock = sock  # Initialize data for thread

    def run(self):
        try:
            while True:
                data, address = self.sock.recvfrom(1024)
                print str(data)
        except socket.error:
            pass
```

Client read config file to determine the IP of server

```
config = {}
execfile("Server.conf", config)
SERVER_ADDR = config["server"]
PORT = 49512
```

Init “incoming thread” to handle incoming message:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_ADDR, PORT))
print ">> Connected to Server"
thread_incoming = IncomingThread(client_socket)
thread_incoming.start()
```

Client read from keyboard, and send to server

```
inp = ''
while inp != 'quit':
    if inp != '':
        client_socket.send(inp)
    inp = raw_input()

client_socket.close()
```