



M4102 – Prog. réseau

~~Année~~ 2015/2016



Programmation d'un mini-chat

Serveur

Le serveur doit :

- **Accepter de multiples clients via la fonction `select`**
- **Conserver une liste des clients et des pseudos**
- **Informer les clients des utilisateurs présents**
- **Gerer les utilisateurs via les messages `USER` et `QUIT`**
- **Relayer les messages**

Dictionnaires

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-

# creation
user_list = {}

# ajout
user_list[socket_a] = 'alice'
user_list[socket_b] = 'bob'

# utilisation
username = user_list[socket_a] # 'alice'
users = user_list.values()
sockets = user_list.keys()

# suppression
del user_list[socket_a]
```

Client

Le client doit :

- Demander le pseudo au lancement
- Envoyer le pseudo au serveur via un **USER**
- Accepter les saisies et les messages serveur :

```
read_s, w_s, e_s = select.select([sys.stdin, socket_t], [], [])
```

- Gérer la saisie de **quit** et envoyer **QUIT** au serveur



La bibliothèque PodSixNet

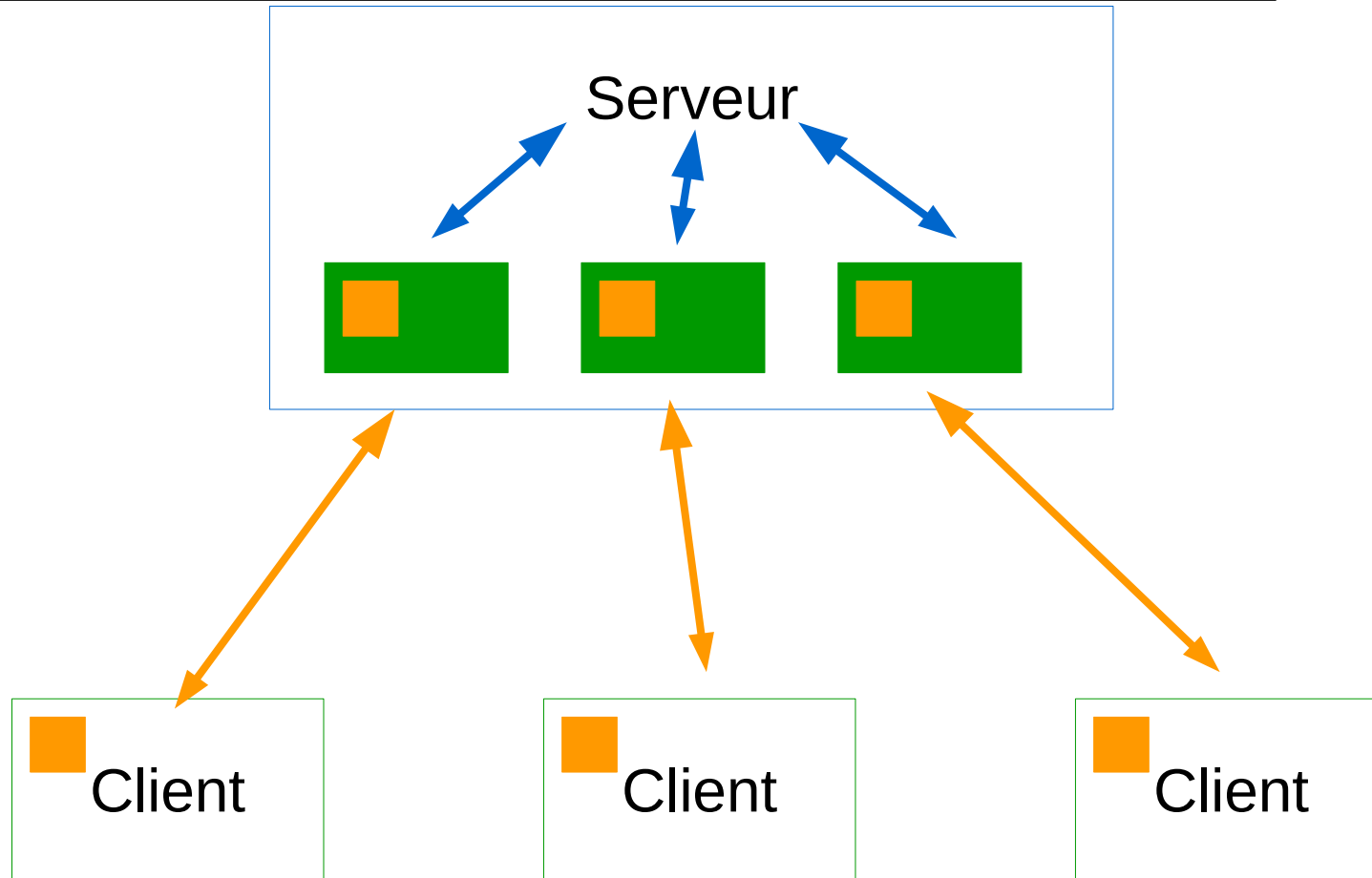
PodSixNet

- MINImaliste
- Orientée jeux vidéos
- Objet
- Callback methods
- Pseudo - protocole

Installation

- PodSixNet sur GitHub
- Download .zip
- Unzip anywhere
- `sudo apt-get install python-setuptools`
- `sudo python2 setup.py install`

Principe général



Principe du client

```
from PodSixNet.Connection import connection, ConnectionListener

class Client(ConnectionListener):
    def __init__(self, host, port):
        self.Connect((host, port))
        username = raw_input('Please enter your pseudo: ')
        connection.Send({"action": "username", "username": username })

    def Loop(self):
        connection.Pump()
        self.Pump()

    def Network(self, data):
        print('message de type % reçu' % data['action'])

    def Network_message(self, data):
        print('message reçu de %s : %s ' % (data['expediteur'], data['message']))

def main_prog():
    while True:
        c.Loop()
        time.sleep(0.001)
```

must do

defines

callback functions

Gestion du réseau

```
|  
### Network event/message callbacks ###  
def Network_connected(self, data):  
    print "You are now connected to the server"  
  
def Network_error(self, data):  
    print 'error:', data['error'][1]  
    connection.Close()  
  
def Network_disconnected(self, data):  
    print 'Server disconnected'  
    sys.exit()
```

events



Principe du serveur

```
class MyServer(Server):
    channelClass = ClientChannel

    def __init__(self, *args, **kwargs):
        Server.__init__(self, *args, **kwargs)
        self.clients = []
        print('Server launched')

    def Connected(self, channel, addr):
        print('New connection')
        self.clients.append(channel)

    def del_client(self, channel):
        print('client %s has quit, removing ...' % channel.username)
        self.clients.remove(channel)

def main_prog():
    my_server = MyServer(localaddr = (sys.argv[1], int(sys.argv[2])))

    while True:
        my_server.Pump()
        time.sleep(0.01)
```

clients list
(channel type)

callback

function

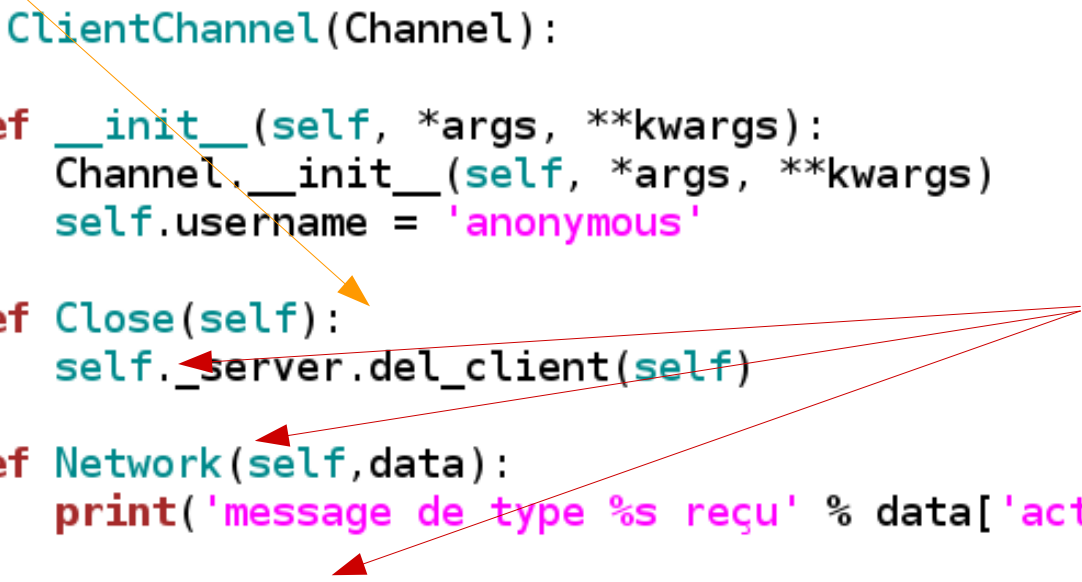
must do

Principe du Serveur (2)

calls server function

```
class ClientChannel(Channel):  
  
    def __init__(self, *args, **kwargs):  
        Channel.__init__(self, *args, **kwargs)  
        self.username = 'anonymous'  
  
    def Close(self):  
        self._server.del_client(self)  
  
    def Network(self, data):  
        print('message de type %s reçu' % data['action'])  
  
    def Network_username(self, data):  
        self.username = data['username']  
        print('Username changed to %s' % data['username'])
```

callback



+ self.Send method

Exercice

Récupérez les programmes *base_client.py* et *base_serveur.py* et complétez-les pour obtenir le fonctionnement suivant :

- Le client se connecte et envoie un message
- A la réception, le serveur affiche et répond,
- A la réception, le client affiche et se ferme

Exercice 2

A partir des programmes *base_client.py* et *base_serveur.py*, recréez le système de « tchat » réalisé précédemment

Pour la saisie utilisateur, vous utiliserez un select non bloquant, en spécifiant une valeur de timeout :

```
read_s, w_s, e_s = select.select([sys.stdin], [], [], 0.01)
```

