Socket

Table des matières

1 Introduction

Utilisation des sockets pour la communication client-serveur.

2 Client

La socket garde la connexion ouverte durant toute la session. Les sockets n'étant pas prévues à cet effet, il faut envoyer un `\n` pour envoyer effectivement les données.

2.1 Telnet

```
$ telnet hostName portNumber
...
Ctrl + ]
> quit
$
```

2.2 Python

La socket ouvre une nouvelle connexion à chaque requette. Elle transmet la réponse puis se ferme.

```
#!/usr/bin/python
import sys
import socket

class ClientSocket():

    def __init__(self, ServerHost, ServerPort):

        self.ServerHost = ServerHost
        self.ServerPort = ServerPort
        self.socket = None


    def SendAndRecv(self, message):
        response = "socket error"
        print "%s\t..." % message,

        try:
```

```
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        socket.setdefaulttimeout(1)
        self.socket.connect((self.ServerHost, self.ServerPort))
        self.socket.send(message + "\n")

        response = ""
        newResponse = self.socket.recv(8)
        while (newResponse != ""):
            response = response + newResponse
            newResponse = self.socket.recv(8)

        # remove ending \n
        response = response[:-1]

    except socket.error, err:
        response = "%s" % (err)

    self.socket.close()
    return response
```

3 Serveur

La solution utilise des fils (au sens fork) plutôts que des fils d'execution (thread) bien que ceux-ci soit plus réactifs.

La solution retenue est d'ouvrir une nouvelle connexion à chaque requette et de fermer la sockette une fois la réponse transmise.

Rien n'a été fait pour gérer la concurrence entre les fils.