Université de Nice Sophia Antipolis
UFR Sciences - Dépt. Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TD n° 4 : Organisation d'un serveur web

L'objectif de ce TD est d'explorer l'utilisation combinée des sockets et de la primitive `select`.

1 Un premier serveur

Expliquez ce que fait le programme suivant et comment on pourrait le tester à partir de la ligne de commande du shell Unix :

```
import select, socket, sys

host = ''
port = 50000
backlog = 5
size = 1024
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host,port))
server.listen(5)
input = [server,sys.stdin]
running = 1
while running:
    inputready,outputready,exceptready = select.select(input,[],[])
    for s in inputready:
        if s == server:
            # handle the server socket
            client, address = server.accept()
            input.append(client)

        elif s == sys.stdin:
            # handle standard input
            junk = sys.stdin.readline()
            running = 0

        else:
            # handle all other sockets
            data = s.recv(size)
            if data:
                s.send(data)
            else:
                s.close()
                input.remove(s)
server.close()
```

2 Un deuxieme serveur

— Meme question que précédemment. Une fois que vous avez bien compris son fonctionnment, modifiez-le de façon à le transformer en serveur `rsh`, c'est-à-dire pour permettre l'exécution d'une commande distante et en reécupérer le résultat.

```
import sys,os,socket,time,select

connsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
connsock.bind(("",8801))
connsock.listen(5)
nb_open = 0
# Create list of potential readers and place connection socket in
# first position
potential_readers = [connsock]
first = True

while first or nb_open > 0:
    first = False
    readers,writers,errors = select.select(potential_readers,[],[],60)
    for sock_ready in readers:
        if sock_ready == connsock:
            conn,(addr,port) = connsock.accept()
            potential_readers.append(conn)
            print "Incoming connexion from %s on port %d..."%(addr,port)
            raw_input("Appuyer sur entree...")
            nb_open+=1
        else:
            msg=sock_ready.recv(1)
            if (len(msg) == 0):
                print "NULL message. Closing connection..."
                sock_ready.close()
                # Remove the closed connection from potential readers
                for i in xrange(1,len(potential_readers)):
                    if potential_readers[i] == sock_ready:
                        potential_readers.pop(i)
                nb_open -= 1
            else: os.write(1,msg)
connsock.close()
print "last connection closed. Bye!"
sys.exit(0)
```

3 Fonctionnement d'un serveur web

Dans les exemples précédents, lorsque le processus est occupé à lire un socket de transfert, il n'est pas disponible pour accepter une connection, et vice-versa. Dans le cas d'un serveur web, le nombre de connections simultanées peut être très élevé. Cela se traduit par une lenteur désagréable, voire inacceptable du serveur. Pour parer à ce problème, une idée classique consiste à construire un serveur multi-processus : au lieu de faire faire tout le travail à un seul processus, on fait appel à plusieurs processus. Par exemple, on peut faire en sorte que seul le processus père accepte les connections, puis qu'il crée un processus fils pour traîter chaque nouvelle connection. Afin de préparer le projet, commencez à réfléchir à un tel algorithme, en faisant bien attention à éviter les zombis ...