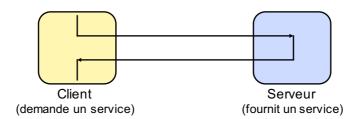
# Communication par sockets

Olivier Dalle
Université de Nice - Sophia Antipolis
http://deptinfo.unice.fr/
D'après le cours original de
Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et MAG-LSR)

http://sardes.inrialpes.fr/~krakowia

### Rappel : le réseau vu de l'utilisateur (1)

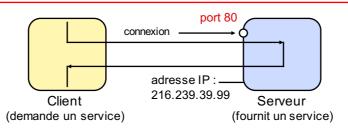


Le schéma client-serveur a été vu en TD pour des processus sur une même machine. Ce schéma se transpose à un réseau, où les processus client et serveur sont sur des machines différentes.

Pour le client, un service est souvent désigné par un nom symbolique (par exemple mail, http://..., telnet, etc.). Ce nom doit être converti en une adresse interprétable par les protocoles du réseau.

La conversion d'un nom symbolique (par ex. http://www.google.com) en une adresse IP (216.239.39.99) est à la charge du service DNS

# Le réseau vu de l'utilisateur (2)



En fait, l'adresse IP du serveur ne suffit pas, car le serveur (machine physique) peut comporter différents services; il faut préciser le service demandé au moyen d'un numéro de port, qui permet d'atteindre un processus particulier sur la machine serveur.

Un numéro de port comprend 16 bits (0 à 65 535). Les numéros de 0 à 1023 sont réservés, par convention, à des services spécifiques. Exemples :

7: echo 23: telnet (connexion à distance) 80:

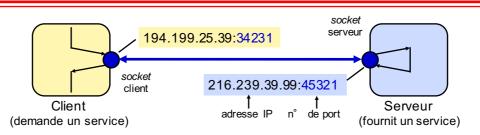
serveur web 25 : mail

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-3

# Le réseau vu de l'utilisateur (3)

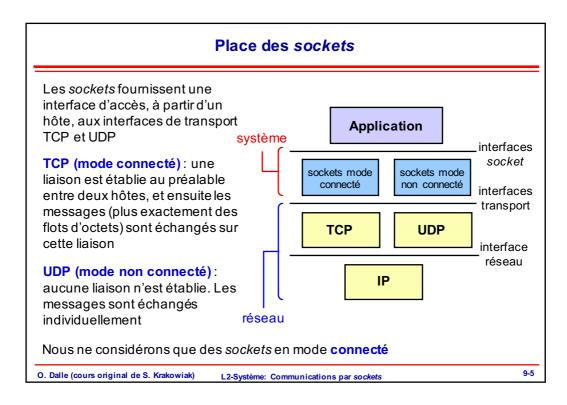


Pour programmer une application client-serveur, il est commode d'utiliser les sockets (disponibles en particulier sous Unix). Les sockets fournissent une interface qui permet d'utiliser facilement les protocoles de transport TCP et UDP

Une socket est simplement un moyen de désigner l'extrémité d'une connexion, côté émetteur ou récepteur, en l'associant à un port. Une fois la connexion (bidirectionnelle) établie via des sockets entre un processus client et un processus serveur, ceux-ci peuvent communiquer en utilisant les mêmes primitives (read, write) que pour l'accès aux fichiers.

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets



### Le protocole TCP

#### Principales caractéristiques de TCP

#### Communication bidirectionnelle par flots d'octets

#### **Transmission fiable**

Fiabilité garantie dès lors que la liaison physique existe

#### Transmission ordonnée

Ordre de réception identique à l'ordre d'émission

#### Contrôle de flux

Permet au récepteur de limiter le débit d'émission en fonction de ses capacités de réception

#### Contrôle de congestion

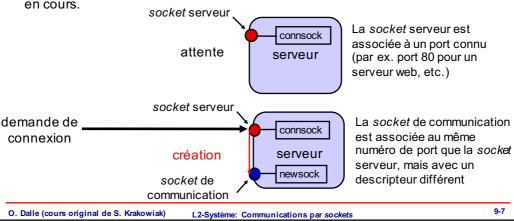
Permet d'agir sur le débit d'émission pour éviter la surcharge du réseau

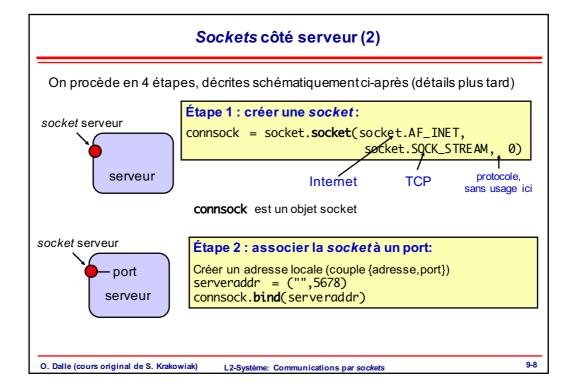
ne pas confondre contrôle de flux (entre récepteur et émetteur ) et contrôle de congestion (entre réseau et émetteur)

# Sockets côté serveur (1)

Un serveur fournit un service à des clients. Il doit donc attendre une demande, puis la traiter.

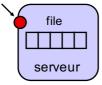
Les fonctions d'attente et de traitement sont séparées, pour permettre au serveur d'attendre de nouvelles demandes pendant qu'il traite des requêtes en cours.





# Sockets côté serveur (3)

#### socket serveur



Étape 3 : indiquer que c'est une socketserveur
QUEUE\_SIZE = 5 # par exemple
connsock.listen(QUEUE\_SIZE)

Taille max de la file d'attente des demandes

Une socket serveur est en attente de demandes de connexion. Si une demande arrive pendant qu'une autre est en cours de traitement, elle est placée dans une file d'attente. Si une demande arrive alors que la file est pleine, elle est rejetée (pourra être refaite plus tard); voir primitive connect plus loin.

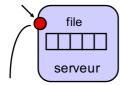
O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-9

# Sockets côté serveur (4)

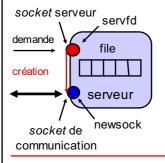
#### socket serveur



**Étape 4a**: **se mettre en attente des demandes** newsock,(addr,port) = **connsock.accept**()

la primitive accept est bloquante

prête à accepter les demandes de connexion



Étape 4b: après acceptation d'une demande newsock,(addr, port) = connsock.accept()

nouvel objet socket
Il sera utilisé pour envoyer ou recevoir des données sur la nouvelle connexion

adresse et port du client qui a initié la nouvelle connexion.

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

### Résumé des primitives pour les sockets côté serveur

```
import socket
sock = socket.socket(domain, type, protocol)
   # crée une socket client ou serveur, renvoie descripteur)
sock.bind((addr,port))
   # associe une socket à une adresse locale
sock.listen(maxqueuesize)
   # déclare une socket comme serveur avec taille max queue
readsock = sock.accept()
   # met une socket serveur en attente de demandes de connexion
```

Il existe en outre une primitive select, traitée plus loin.

O. Dalle (cours original de S. Krakowiak)

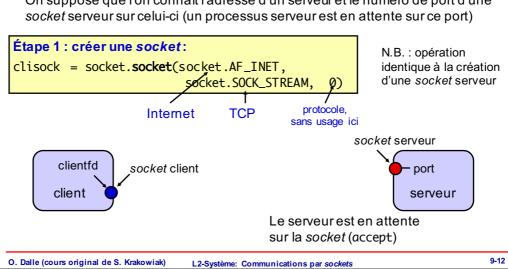
L2-Système: Communications par sockets

9-11

# Sockets côté client (1)

On procède en 2 étapes, décrites schématiquement ci-après

On suppose que l'on connaît l'adresse d'un serveur et le numéro de port d'une



# Sockets côté client (2)

# Étape 2 : établir une connexion entre la socket client et le serveur

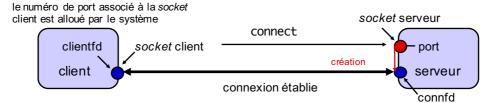
addr = ("www.unice.fr",80)

# remplir serveraddr avec adresse et n° port du serveur

clisock.connect(addr)

# renvoie 0 si succès, 1 si échec

connect envoie une demande de connexion vers la socket serveur



Le client et le serveur peuvent maintenant dialoguer sur la connexion

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-13

### Résumé des primitives pour les sockets côté client

```
import socket
```

clisock = socket.socket(domain, type, protocol);
 # crée une socket client ou serveur, renvoie descripteur)

clisock.connect((addr,port))

# envoie une demande de connexion à un serveur sur un

# port donné

# Échanges sur une connexion entre sockets

Une fois la connexion établie, le client et le serveur disposent chacun d'un object socket vers l'extrémité correspondante de la connexion. Cet objet fonctionne de façon un peu similaire à un descripteur de fichier : on peut l'utiliser pour les méthodes recv et send ; on le ferme avec close.



O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-15

# Un application client-serveur avec sockets (1)

Principes de la programmation d'une application avec sockets (les déclarations sont omises).

#### Côté serveur:

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

# Un application client-serveur avec sockets (2)

Principes de la programmation d'une application avec sockets (les déclarations sont omises).

#### Côté client:

```
clisock = socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)
clisock.connect(("www.unice.fr",port))
  # ouvre une connexion vers le serveur
client_prog(clisock)
  # envoyer requêtes et recevoir réponses avec clientfd
  # en utilisant recv et send

# à la fin des échanges, fermer le descripteur
clisock.close()
```

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-17

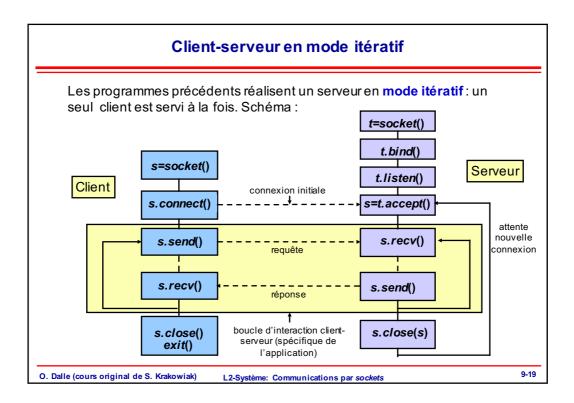
### Un application client-serveur avec sockets (3)

#### Pour exécuter l'application :

Lancer le programme serveur sur une machine, en indiquant un numéro de port (>1023, les numéros ≤1023 sont réservés); de préférence en travail de fond

Lancer le programme client sur une autre machine (ou dans un autre processus de la même machine), en spécifiant adresse du serveur et numéro de port

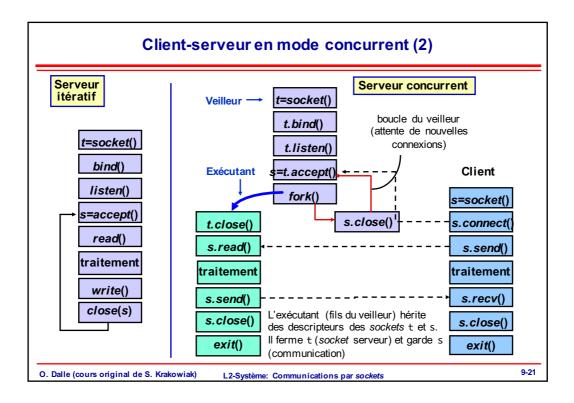
N.B. On n'a pas prévu d'arrêter le serveur (il faut tuer le processus qui l'exécute). Dans une application réelle, il faut prévoir une commande pour arrêter proprement le serveur



# Client-serveur en mode concurrent (1)

Pour réaliser un serveur en **mode concurrent**, une solution consiste à créer un nouveau processus pour servir chaque demande de connexion, le programme principal du serveur ne faisant que la boucle d'attente sur les demandes de connexion.

Donc il y a un processus principal (appelé *veilleur*) qui attend sur accept(). Lorsqu'il reçoit une demande de connexion, il crée un processus fils (appelé *exécutant*) qui va interagir avec le client. Le veilleur revient se mettre en attente sur accept(). Plusieurs exécutants peuvent exister simultanément.



### La primitive select.select

- Par défaut l'appel à read ou recv est bloquant
  - S'il n'y a rien a lire, on attend indéfiniment...
- Comment lire et recevoir de plusieurs sources en meme temps ?
  - ◆ Par exemple attendre en même temps:
    - soit une entrée au clavier => lire fd=0
    - ♦ soit l'arrivée d'un message sur un socket connecté =>lire listensœk
    - soit l'arrivée d'une demande de connection => lire connsock
    - \*
  - ◆ **Pb:** si je me bloque en attendant sur fd=0 (stdin=clavier), je ne peux pas savoir ce qui se passe sur listensock ou connsock
- Solutions
  - Serveur concurrent
  - select.select

# Exemple d'utilisation de select.select

O. Dalle (cours original de S. Krakowiak)

L2-Système: Communications par sockets

9-23

### Résumé de la séance 9

- Communication par sockets en mode connecté
  - Principe
  - Primitives socket, bind, listen, accept, connect
  - Utilisation des primitives
- Programmation client-serveur avec sockets
  - ♦ Serveur itératif
  - ♦ Serveur à processus concurrents
- Select