Architecture client/serveur

Modèle Client-Serveur

Repose sur une communication d'égal à égal entre les applications ; communication réalisée par dialogue entre processus deux à deux

- un processus client
- un processus serveur

• Les processus ne sont pas identiques mais forment plutôt un système coopératif se traduisant par un échange de données

– le client réceptionne les résultats finaux délivrés par le serveur

- Le client initie l'échange
- Le serveur est à l'écoute d'une requête cliente éventuelle
- Le service rendu = traitement effectué par le serveur

Serveur

- **Un programme serveur**
  - tourne en permanence, attendant des requêtes
  - peut répondre à plusieurs clients en même temps
- **Nécessite**

– une machine robuste et rapide, qui fonctionne une machine robuste et rapide qui fonctionne 24h/24

  - alimentation redondante, technologie RAID ...

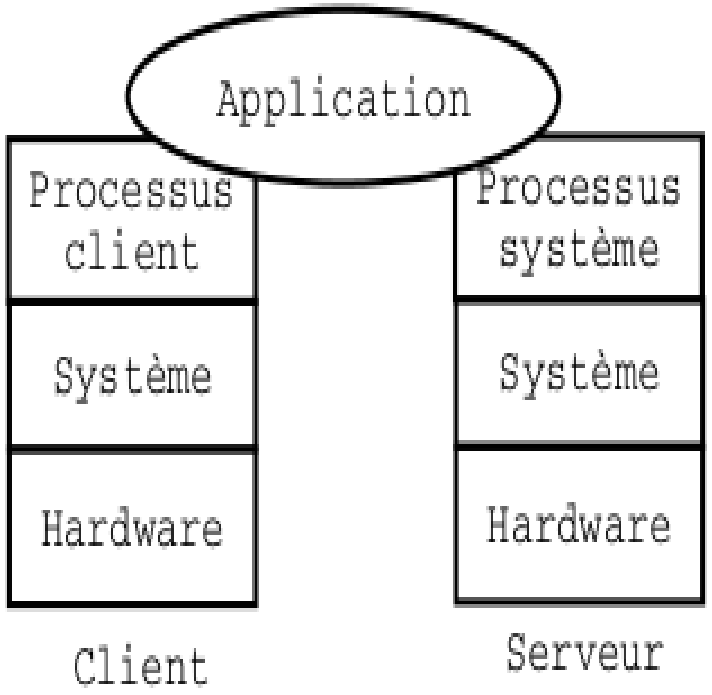
– la présence d'administrateurs systèmes et réseau pour gérer les serveurs

Exemples de serveurs

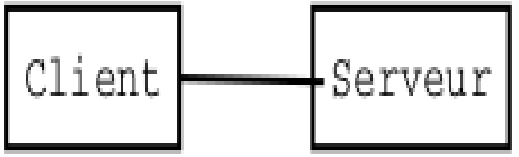
- Serveur de fichiers (NFS, SMB)
- Serveur d'impression (lpd, CUPS)
- Serveur de calcul
- Serveur d'applications
- Serveur de bases de données
- Serveur de temps
- Serveur de noms (annuaire des services)

Types d'architecture client-serveur

Le modèle :

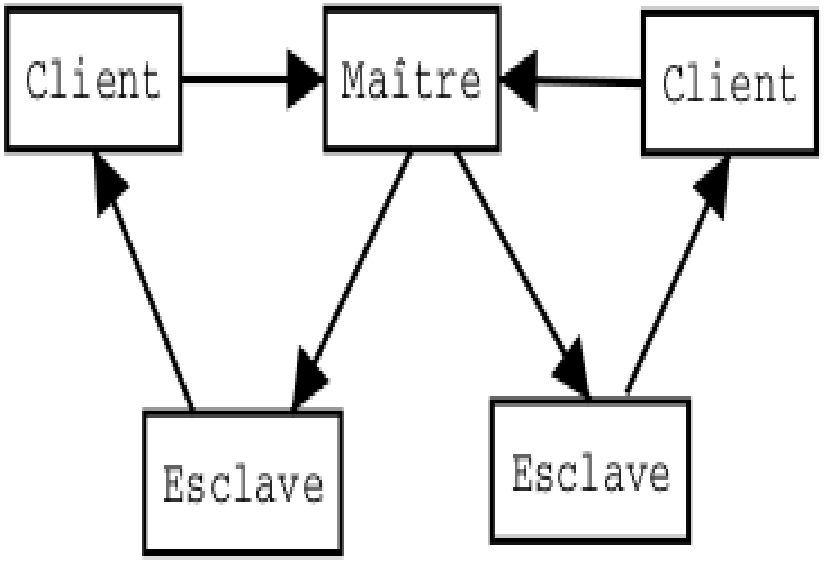


Un client, un serveur :



Un client, plusieurs serveurs :

Plusieurs clients, un serveur :



Le modèle client / serveur

- Le client demande l'exécution d'un service
- Le serveur réalise le service
- Client et serveur sont généralement localisés sur deux machines distinctes

Le modèle client / serveur

- **Communication par messages**

– **Requête**

Requête : paramètres d'appel, spécification du service requis
(message transmis par un client à un serveur
décrivant l'opération à exécuter)

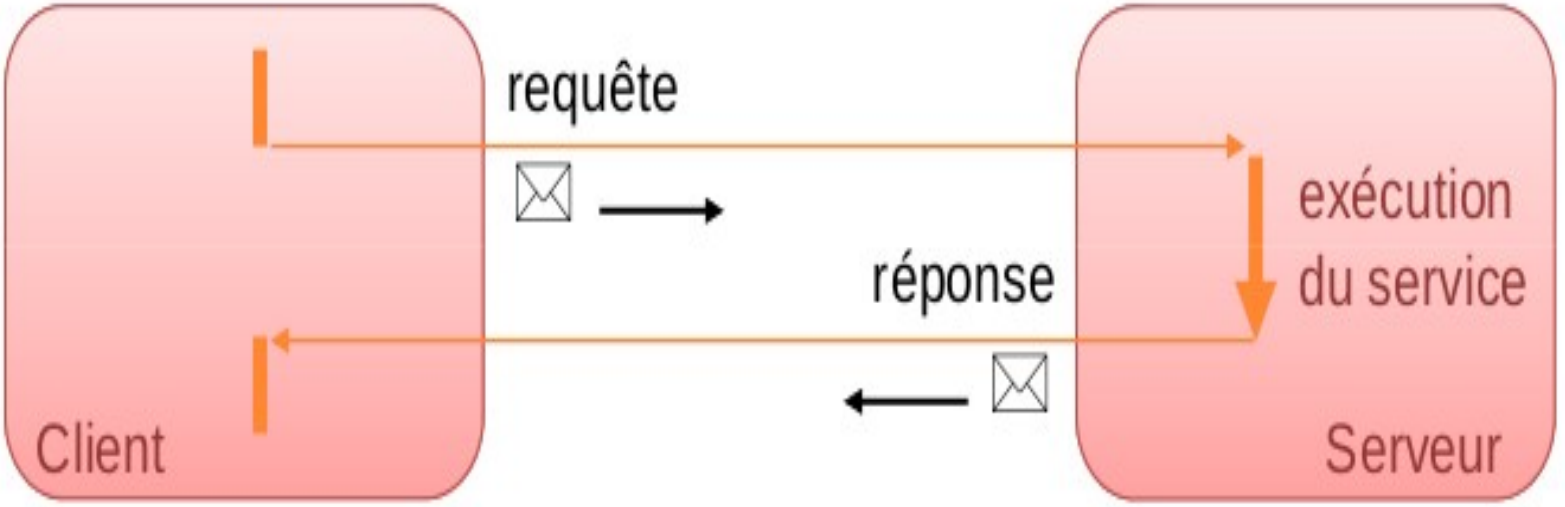
– **Réponse :** résultats, indicateur éventuel d'exécution ou d'erreur
(message transmis par un serveur à un client suite à
l'exécution d'une opération, contenant le résultat de
l'opération)

Le modèle client / serveur

Mode de communication synchrone

Le client envoie la requête et attend la réponse.

Emissio bloquante(dans le modèle de base le client est bloqué en attente de la réponse)



Modèle de communication par messages

Une application produit des messages (producteur) et une application les consomme (consommateur)

Le producteur (l'émetteur) et le consommateur (récepteur) ne communiquent pas directement entre eux mais utilisent un objet de communication intermédiaire (boîte aux lettres ou file d'attente)

Communication asynchrone :

- **Les deux composants n'ont pas besoin d'être connectés en même temps grâce au système de file d'attente**
- **Emission non bloquante: l'entité émettrice émet son message, et continue son traitement sans attendre que le récepteur confirme l'arrivée du message.**
- **Le récepteur récupère les messages quand il le souhaite.**

Modèle de communication par messages

- Adapté à un système réparti dont les éléments en interaction sont faiblement couplés :

  éloignement géographique des entités communicantes

  - possibilité de déconnexion temporaire d'un élément

- **Deux modèles de communication :**

Point à point: Les messages ne sont lus que par un seul consommateur.

Une fois qu'un message est lu, il est retiré de la file d'attente.

Multi-points:

Le message est diffusé à tous les éléments d'une liste de Destinataires

- **Exemple: messagerie électronique**

Modèle de communication par événements

Concepts de base

Evénement = changement d'état survenant de manière asynchrone (par rapport à ses “clients”)

Réaction = exécution d’une opération prédéfinie liée à l’événement

Mode de communication asynchrone et anonyme

indépendance entre l’émetteur (producteur) et les destinataires (consommateurs) d’un Événement

Modèle Publish/Subscribe (par abonnement):

les applications consommatrices des messages s'abonnent à un topic (sujet)

Les messages envoyés à ce topic restent dans la file d'attente jusqu'à ce que toutes les applications abonnées aient lu le message

Modèle de communication par événements

Deux modes de consommation des messages:

«Mode Pull» - réception explicite : les clients viennent prendre régulièrement leurs messages

«Mode Push» - délivrance implicite : une méthode prédéfinie est attachée à chaque type de message et elle est appelée automatiquement à chaque occurrence de l'évènement.

la réception d'un événement entraîne l'exécution de la réaction associée

Exemple : surveillance des systèmes (changement d'états de configuration, alertes, statistiques

Gestion des processus

Client et serveur exécutent des processus distincts:

– Le client est suspendu lors de l'exécution de la requête (appel synchrone)

– Plusieurs requêtes peuvent être traitées par le serveur

- mode itératif
- mode concurrent

Gestion des processus

- Mode de gestion des requêtes
  - **itératif**
- le processus serveur traite les requêtes les unes après les autres
  - **concurrent** basé sur
- **parallélisme réel**
  - système multiprocesseurs par exemple
- **pseudo-parallélisme**
  - schéma veilleur-exécutants
- la concurrence peut prendre plusieurs formes
  - plusieurs processus (un espace mémoire par processus)
  - plusieurs processus légers (threads) dans le même espace mémoire

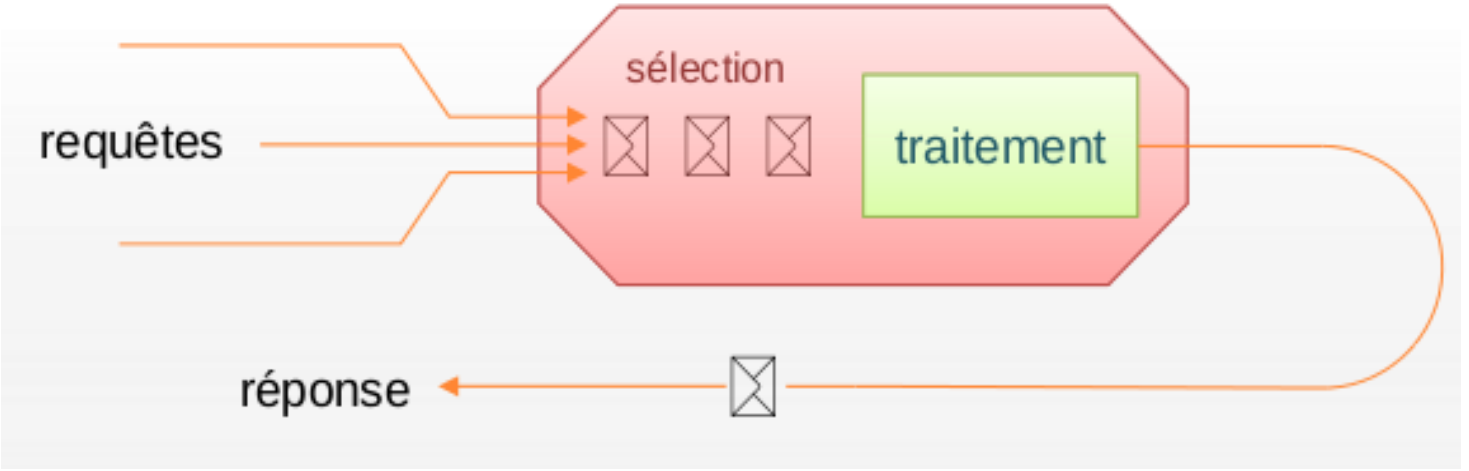
Gestion des processus dans le serveur

- **Processus serveur unique**

```
while (true) {

receive ( (client_id,message)

extract (message, service_id, params)
do_service[service_id] (params, results)
send (client_id, results)
}
```



Gestion des processus dans le serveur

- **Schéma veilleur-exécutants**

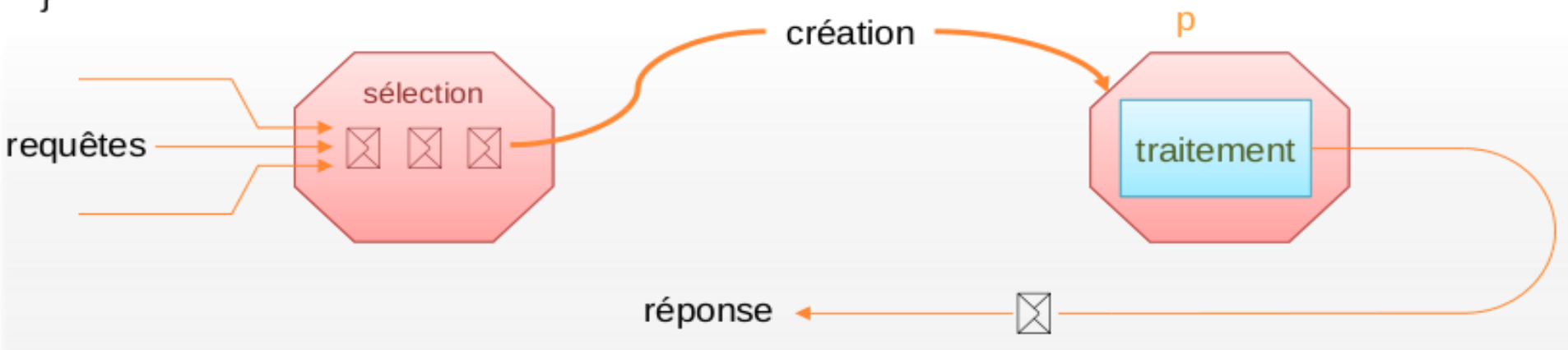
Processus veilleur

```
while (true) {
  receive (client_id, message)
  extract (message, service_id, params)
  p = create_thread (client_id, service_id,
      params)
}
```

Création dynamique des exécutants

programme de p

```
do_service[service_id] (params, results)
send (client_id, results)
exit
```



Mise en œuvre du modèle client / serveur

• Besoin d'un support pour transporter les informations entre le client et le serveur

– **Bas niveau**

• Utilisation directe du transport : sockets((construits sur TCP ou UDP)

– **Haut niveau**

• Intégration dans le langage de programmation : RPC ou Remote Procedure Call (construits sur sockets)

• Nécessité d'établir un protocole entre le client et le serveur pour qu'ils se comprennent

Les sockets

Qu'est ce qu'un socket?

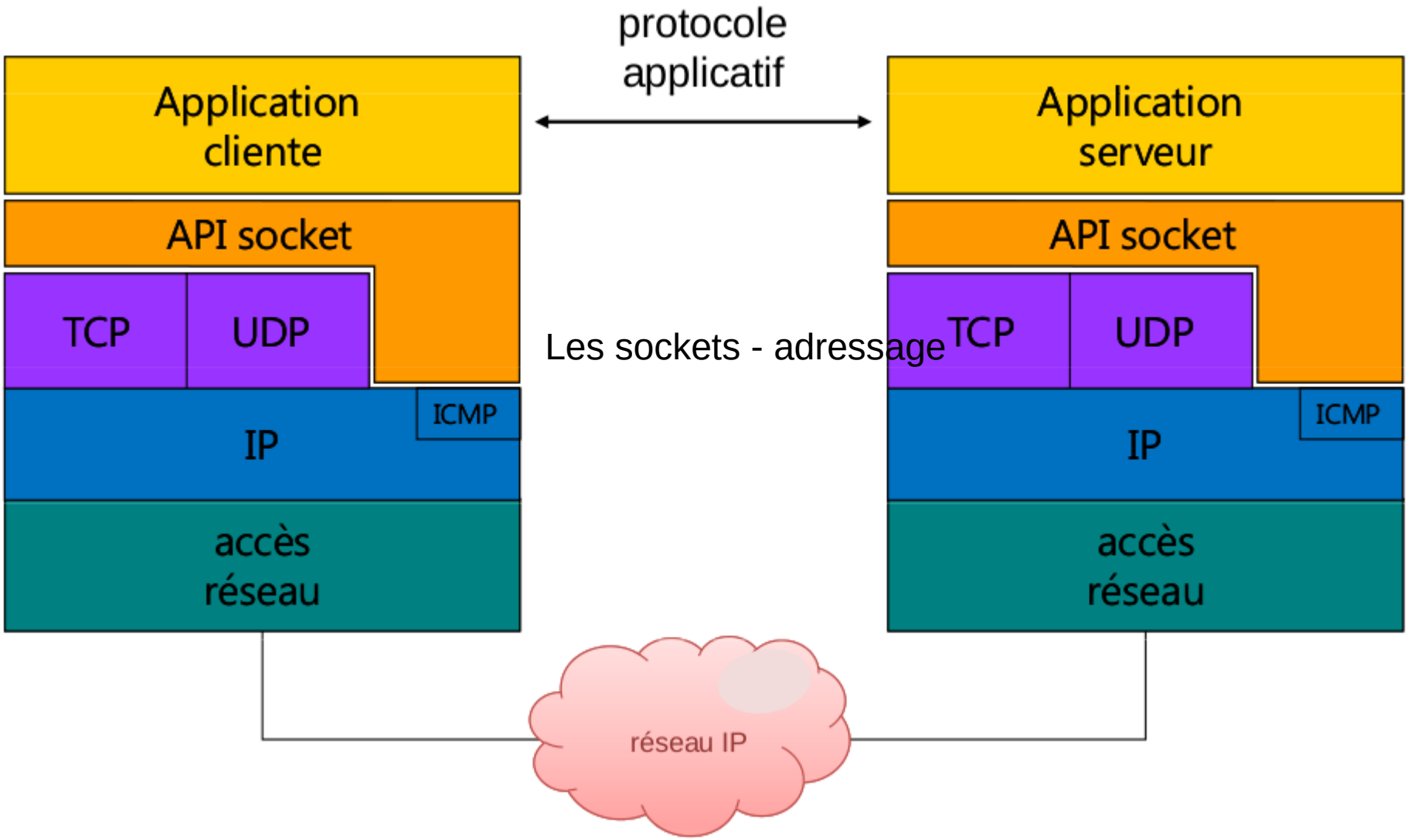
- API (Application Program Interface) socket

– mécanisme d'interface de programmation

permet aux programmes d'échanger des données

- les applications client/serveur ne voient les couches de communication qu'à travers l'API socket (abstraction)
- N'implique pas forcément une communication par le réseau

L'API socket



Les sockets - adressage

L'API socket

Deux processus communiquent en émettant et recevant des données via les sockets,

Les sockets sont des portes d'entrées/sorties

vers le réseau (la couche transport)

Une socket est identifiée par une adresse de

transport qui permet d'identifier les processus de

l'application concernée

Une adresse de transport = un numéro de port

(identifie l'application) + une adresse IP

(identifie le serveur ou l'hôte dans le réseau)

Notion de port

- Un service rendu par un programme serveur sur une machine est accessible par un port
- Un port est identifié par un entier (16 bits)

– de 0 à 1023

  - ports reconnus ou réservés
  - sont assignés par l'IANA (Internet Assigned Numbers Authority)
  - donnent accès aux services standard : courrier (SMTP port 25), serveur web (HTTP port 80) ...

> 1024

  - ports « utilisateurs » disponibles pour placer un service applicatif quelconque
  - Un service est souvent connu par un nom (FTP, …)
    - La correspondance entre nom et numéro de port est donnée par le fichier /etc/services

L'API socket

Avec les protocoles UDP et TCP, une connexion est entièrement définie sur chaque machine par :

- le type de protocole (UDP ou TCP)
- l'adresse IP
- le numéro de port associé au processus
  - serveur : port local sur lequel les connexions sont attendues
  - client : allocation dynamique par le système

L'API socket

Différents types de sockets

Stream Sockets (TCP)

établir une communication en mode connecté

si connexion interrompue : applications informées

Datagram Sockets (UDP)

établir une communication en mode non connecté

données envoyées sous forme de paquets

indépendants de toute connexion. Plus rapide, moins fiable que TCP

Mode connecté / non connecté

- Deux réalisations possibles
  - Mode connecté (protocole TCP)
  - Mode non connecté (protocole UDP)
- **Mode connecté**
  - Ouverture d'une liaison, suite d'échanges, fermeture de la liaison
  - Le serveur préserve son état entre deux requêtesGaranties de TCP : ordre, , contrô de flux, , fiabilité
- Adapté aux échanges ayant une certaine durée

(plusieurs messages)

Mode connecté / non connecté

- **Mode non connecté**

– Les requêtes successives sont indépendantes

– Pas de préservation de l'état entre les requêtes

– Le client doit indiquer son adresse à chaque requête (pas de liaison permanente)

– Pas de garanties particulières (UDP)

• gestion de toutes les erreurs à la main : il faut réécrire la couche transport !!!

– Adapté

• aux échanges brefs (réponse en 1 message)

• pour faire de la diffusion

Principes de fonctionnement (mode conc rrent)

Le serveur crée une «socket serveur » (associée à un port) et se met en attente

1) Le client se connecte à la socket serveur

Deux sockets sont alors créés:

- Une « socket client » côté client
- Une « socket service client » côté serveur

Ces sockets sont connectées entre elles

Le client et le serveur communiquent par les sockets

-L'interface est celle des fichiers (read, write)

- La socket serveur peut accepter de nouvelles connexions