المدرسة الدولية الخاصة للتقنيات
POLYTECH INTL

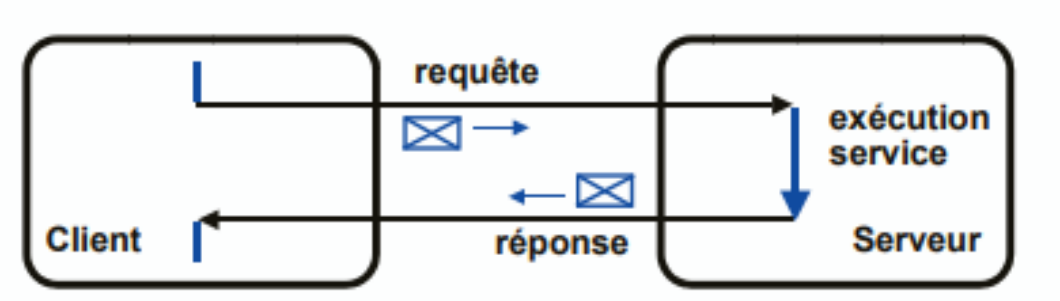
Informatique Répartie
- Sockets -

Ichrak MEHREZ
(m_ichrak@hotmail.fr)

2018-2019

Rappel: Modèle Client/Serveur

- Appel synchrone requête-réponse



- Mise en oeuvre
  - Haut niveau : intégration dans un langage de programmation : ***RPC*** (construit sur sockets)
  - Bas niveau : utilisation directe du transport : sockets (construit sur TCP ou UDP)

Sockets

- **Socket :** mécanisme de communication permettant d'utiliser l'interface de transport (TCP-UDP).

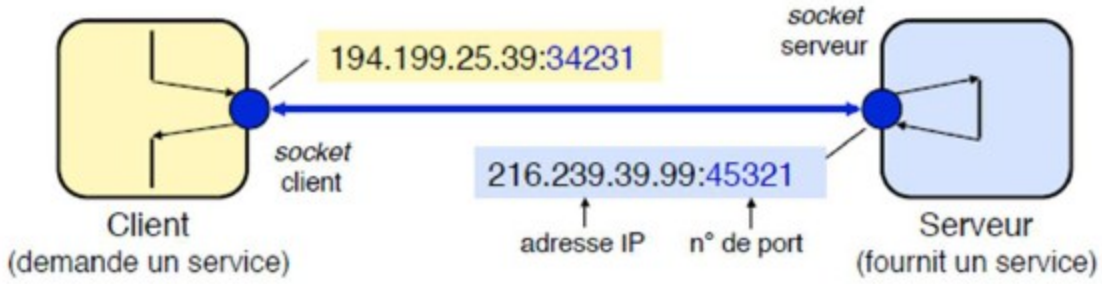
- Introduit dans Unix dans les années 80 ; standard aujourd'hui

- Une socket est un moyen de désigner l'extrémité d'une connexion, côté émetteur ou récepteur, en l'associant à un port.

Sockets: Principe de fonctionnement

3 phases - illustration ci-dessous avec TCP

1. Le serveur crée une "*socket serveur*" (associée à un port) et se met en attente
2. Le client se connecte à la socket serveur ; deux sockets sont alors crées : une "*socket client*", côté client, et une "*socket service client*" côté serveur. Ces sockets sont connectées entre elles
3. 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



Modes de connexion

Mode connecté (protocole TCP)

- Ouverture d'une liaison, suite d'échanges, fermeture de la liaison
- Le serveur préserve son état entre deux requêtes
- Garanties de TCP : ordre, contrôle de flux, fiabilité
- Adapté aux échanges ayant une certaine durée (plusieurs messages)

Mode non connecté (protocole UDP)

- 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)
- Adapté aux échanges brefs (réponse en 1 message)

Modes de connexion

TCP et UDP: Points communs

- Le client a l'initiative de la communication ; le serveur doit être à l'écoute
- Le client doit connaître la référence du serveur [adresse IP, n° de port] (il peut la trouver dans un annuaire si le serveur l'y a enregistrée au préalable, ou la connaître par convention : n° de port préaffféctés)
- Le serveur peut servir plusieurs clients (1 thread unique ou 1 thread par client)

Mode connecté (TCP)

Caractéristiques:

- Etablissement préalable d'une connexion: le client demande au serveur s'il accepte la connexion
- Après initialisation, le serveur est « *passif* » : il est activé lors de l'arrivée d'une demande de connexion du client
- Un serveur peut répondre aux demandes de service de plusieurs clients : les requêtes arrivées et non traitées sont stockées dans une file d'attente

Mode connecté (TCP)

Contraintes

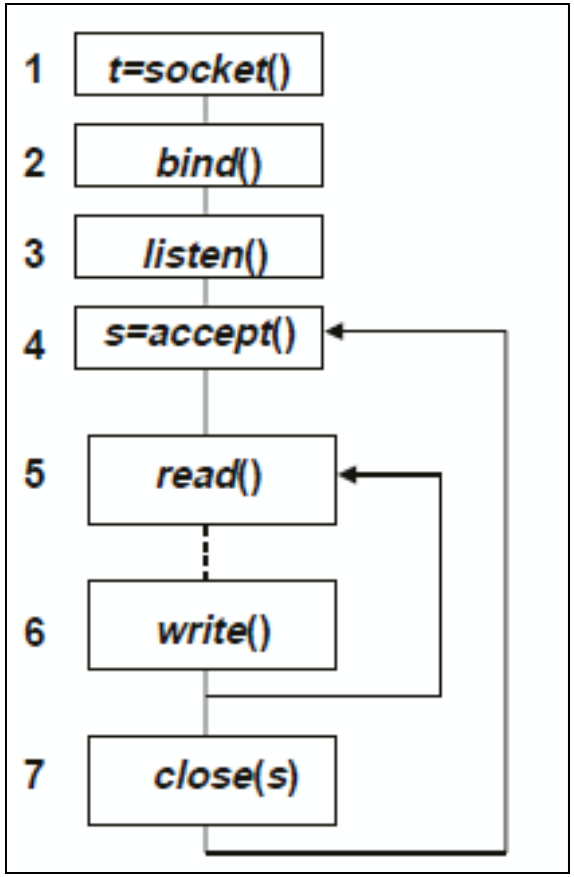
- le client doit avoir accès à l'adresse du serveur (adresse IP, n° de port)

Modes de gestion des requêtes

- itératif : le processus traite les requêtes les unes après les autres
- concurrent : par création de processus fils pour les échanges de chaque requête (ouverture de connexion)

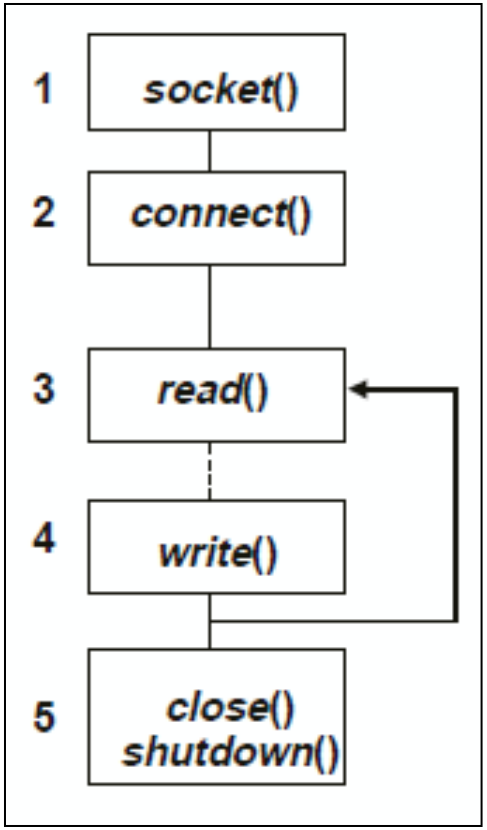
Mode connecté: algorithme d'un serveur

- 1. Création de la socket serveur
- 2. Récupération de l'adresse IP et du numéro de port du serveur ; lien de la socket à l'adresse du serveur
- 3. Mise en mode passif de la socket : elle est prête à accepter les requêtes des clients
- 4. (opération bloquante) : acceptation d'une connexion d'un client et création d'une socket service client, dont l'identité est rendue en retour
- 5. et 6. Lecture, traitement et écriture (selon algorithme du service)
- 7. Fermeture et remise en attente

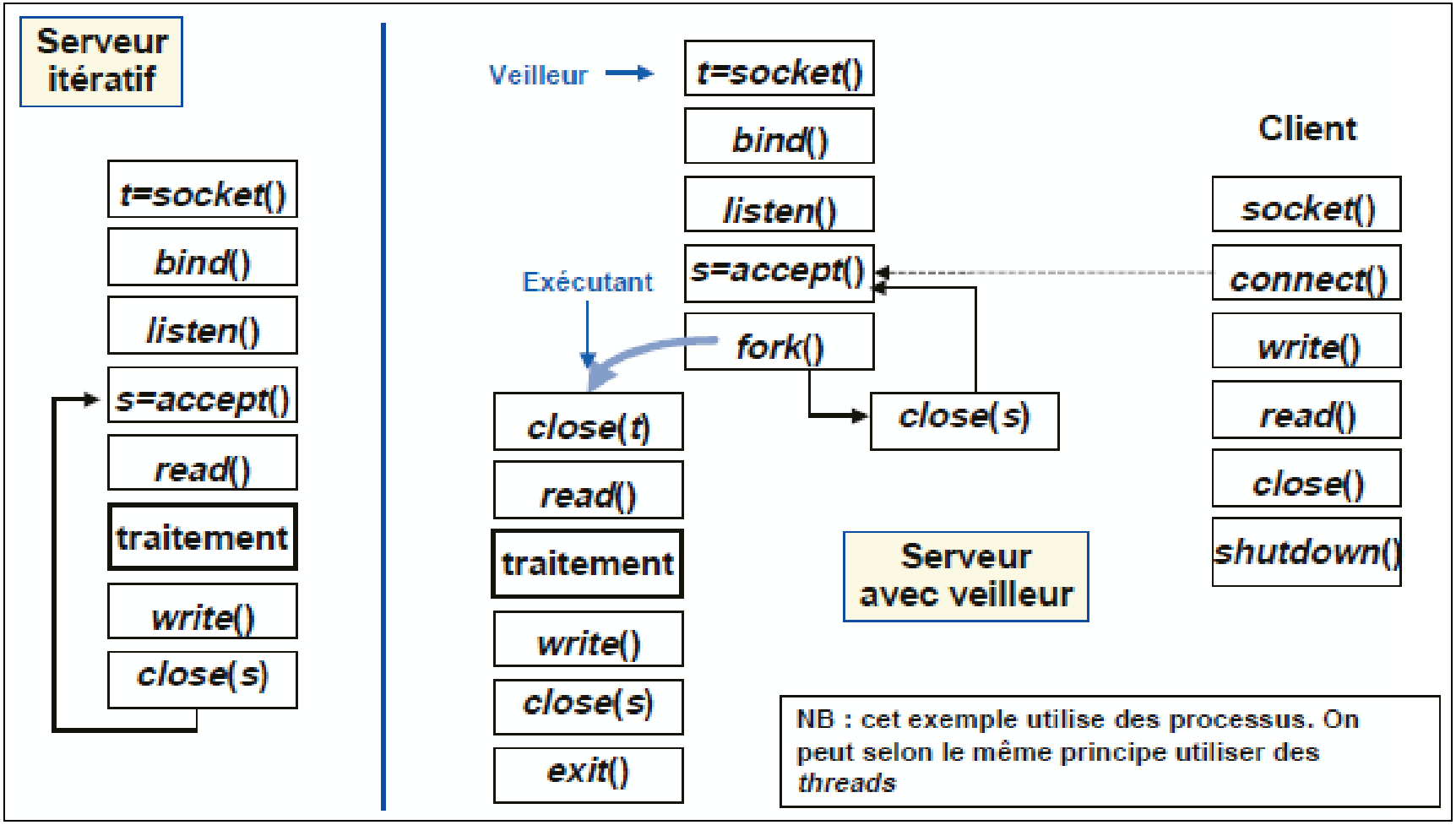


Mode connecté: algorithme d'un client

- 1. Création de la socket
- 2. Connexion de la socket au serveur
  - choix d'un port libre pour la socket par la couche TCP
  - attachement automatique de la socket à l'adresse (IP machine locale + n° de port)
  - connexion de la socket au serveur en passant en paramètre l'adresse IP et le n° de port du serveur
- 3. et 4. Dialogue avec le serveur (selon algorithme du service)
- 5. Fermeture de la connexion avec le serveur



Mode connecté: Gestion des processus



Mode connecté: Exemple en Java

Deux classes de base

- ***ServerSocket*** : socket côté serveur (attend connexions et crée socket service client)

- ***Socket*** : sockets ordinaires, pour les échanges. Fournissent des classes *InputStream* et *OutputStream* (flots d'octets) pour échanger de l'information.

Mode connecté: Exemple en Java

Sur une machine client

doit être connu du client

Sur la machine serveur (ex : "goedel.imag.fr")

```
mySocket = new Socket("goedel.imag.fr", 7654);
```

```
serverSocket = new ServerSocket(7654);
clientServiceSocket = serverSocket.accept();
```

Maintenant le client et le serveur sont connectés

```
PrintWriter out = new PrintWriter(
        mySocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
        new InputStreamReader(
            mySocket.getInputStream()));
```

```
PrintWriter out = new PrintWriter(
    clientServicetSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        clientServiceSocket.getInputStream()));
```

Maintenant le client et le serveur peuvent communiquer via les canaux

```
BufferedReader stdIn = new BufferedReader(
       new InputStreamReader(System.in));
String request; String reply;

while (true) {
    request = stdIn.readLine(); // l'utilisateur entre la requête
    out.println(request);      // envoyer la requête au serveur
    reply = in.readLine()    // attendre la réponse
   System.out.println(reply); // imprimer la réponse
   }
```

```
String request;
String reply;

while (true) {
    request = in.readLine()
    // executer le service
    // traiter  request pour fournir reply
    out.println(reply);
    }
```

Mode non connecté (UDP)

Caractéristiques:

- Pas d'établissement préalable d'une connexion
- Pas de garantie de fiabilité
- Adapté aux applications pour lesquelles les réponses aux requêtes des clients sont courtes (1 message)
- le récepteur reçoit les données selon le découpage effectué par l'émetteur

Mode non connecté (UDP)

Contraintes:

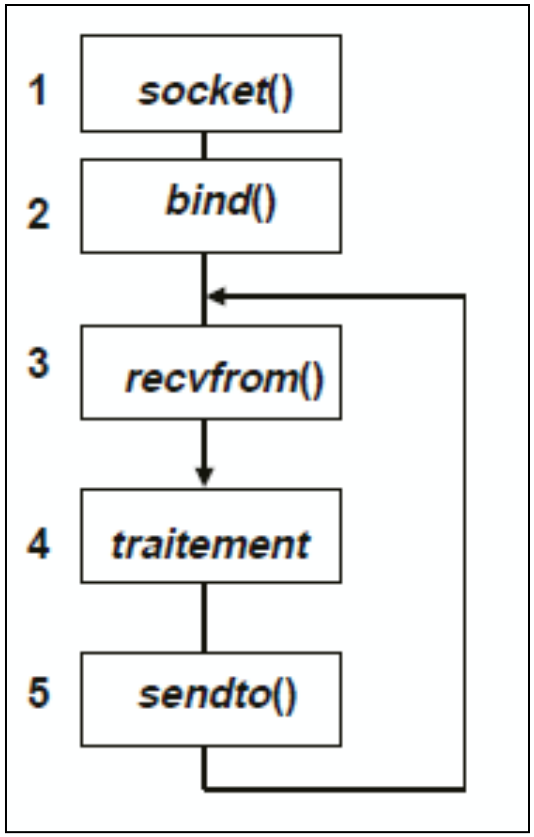
- Le client doit avoir accès à l'adresse du serveur (*adr. IP, port*)
- Le serveur doit récupérer l'adresse de chaque client pour lui répondre (primitives *sendto, recvfrom*)

Mode de gestion des requêtes:

- itératif (requêtes traitées l'une après l'autre)
- concurrent (1 processus ou thread par client)

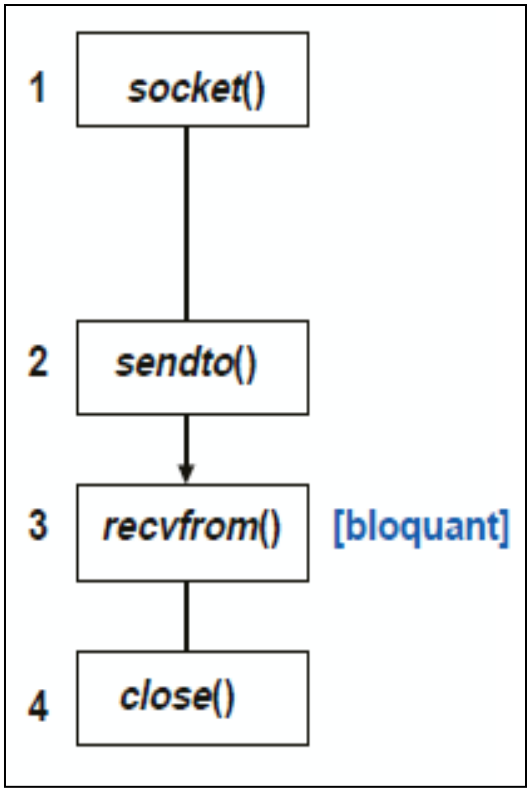
Mode non connecté: algorithme d'un serveur

- 1. Création de la socket
- 2. Récupération de l'adresse IP et du numéro de port du serveur ; lien de la socket à l'adresse du serveur
- 3. Réception d'une requête de client
- 4. Traitement de la requête; préparation de la réponse
- 5. Réponse à la requête en utilisant la socket et l'adresse du client obtenues par *recvfrom* ; retour pour attente d'une nouvelle requête

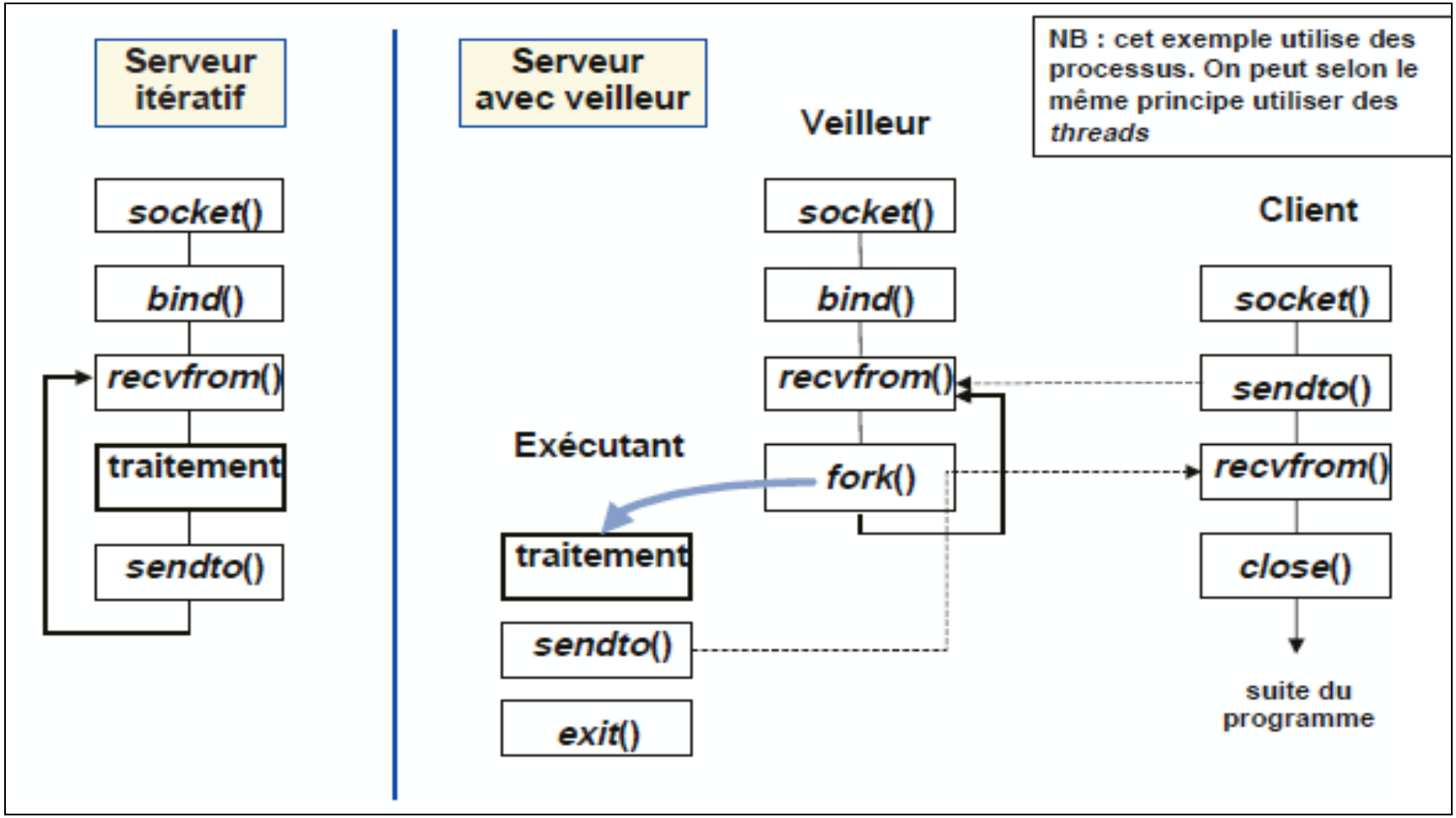


Mode non connecté: algorithme d'un client

- 1. Création de la socket (l'association à une adresse locale [*adresse IP* + *n°port*] est faite automatiquement lors de l'envoi de la requête)
- 2. Envoi d'une requête au serveur en spécifiant son adresse dans l'appel
- 3. Réception de la réponse à la requête
- 4. Fermeture de la socket



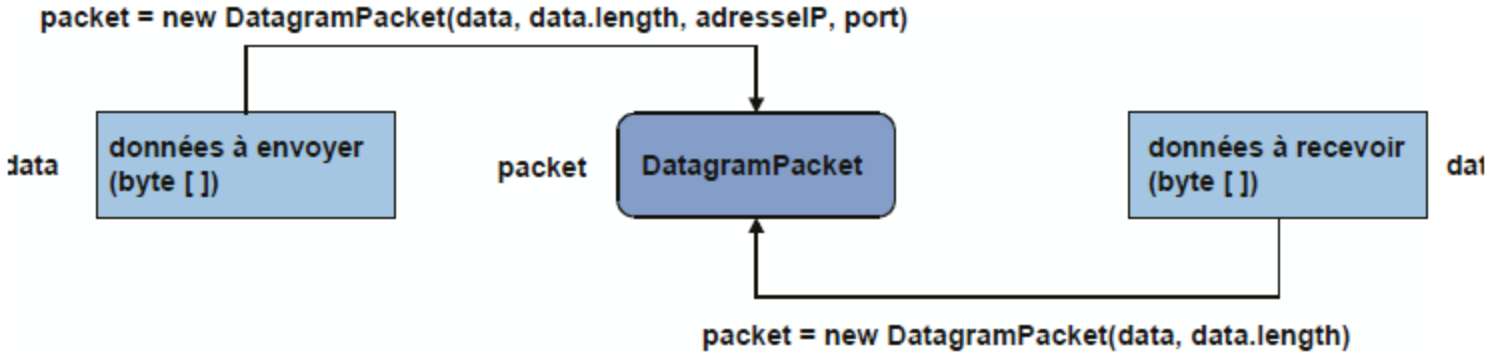
Mode non connecté: Gestion des processus



Mode non connecté: Exemple en Java

Deux classes : *DatagramSocket* et *DatagramPacket*

- ***DatagramSocket*** : un seul type de socket
- ***DatagramPacket*** : format de message pour UDP
  - ➔ Conversion entre données et paquet (dans les 2 sens)



Utilisation

- Échanges simples (question-réponse)
- Messages brefs
- « Streaming » temps réel (performances)

Mode non connecté: Exemple en Java

Sur une machine client

```
DatagramSocket socket = new DatagramSocket();

byte [ ] sendBuffer = new byte [1024] ;
byte [ ] receiveBuffer = new byte [1024] ;
String request, reply;

request = …  // dépend de l'application
sendBuffer = request.getBytes() ;
DatagramPacket outpacket =
   new DatagramPacket (sendBuffer,
                       sendBuffer.length,

                       InetAddress.getbyname("goedel.imag.fr"),
                       7654) ;
socket.send(outpacket) ;

DatagramPacket inpacket =
   new DatagramPacket (receiveBuffer,
                                receiveBuffer.length);
socket.receive(inpacket );
reply = new String(inpacket.getData());
```

doit être connu du client

Envoi requête

Réception réponse

Sur la machine serveur (ex : "goedel.imag.fr")

```
DatagramSocket socket =
       new DatagramSocket(7654);

byte [ ] receiveBuffer = new byte [1024] ;
byte [ ] sendBuffer = new byte [1024] ;
String request, reply;

while (true) {
   DatagramPacket inpacket =
      new DatagramPacket (receiveBuffer,
                                  receiveBuffer.length);
   socket.receive(inpacket ) ;
   request =
         new String(receiveBuffer.getData);
   // déterminer adresse et port du client
   InetAddress clentAddress = inpacket.getAddress(
   int clienPort = inpacket.
   // executer service : traiter request pour fournir re
   sendBuffer = reply.getBytes() ;
   DatagramPacket outpacket =
      new DatagramPacket (sendBuffer,
         sendBuffer.length,  clientAddress, clienPort);
    socket.send(outpacket);
   }
```

Réception requête

Envoi réponse