(Archi4)
Architecture 4
Programmation répartie
Accès concurrentiels

Lecture *Architecture 4*
Programmation répartie 19 janvier 2020

Miklós MOLNÁR

contacter
molnar@lirmm.fr
Institut Universitaire de Technologie de Montpellier

Client/serveur en Java à l'aide des sockets

Miklós MOLNÁR

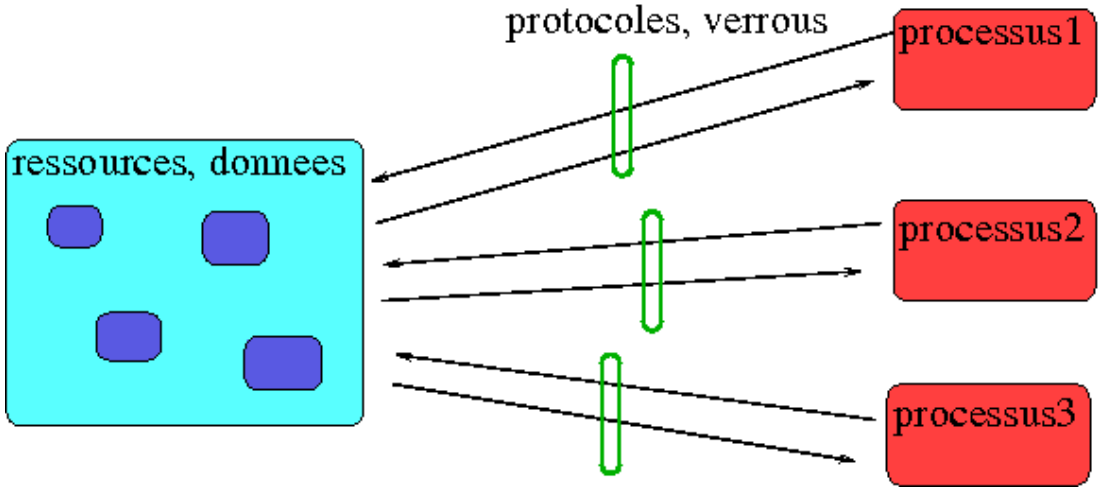
contacter
molnar@lirmm.fr

On suppose que les objets / processus sur des machines différentes ont besoin des données et des traitements centralisés et confiés à un serveur.
On va analyser :

- Modèle client/serveur
- Communication entre processus distants
- Prérequis :
  - Fonctionnement des réseaux
    - Modèle en couches
  - Identification et adressages
    - Adresse MAC > > Adresse IP > > port TCP > > services, ressources
  - Sockets
    - Communication (adresse IP / port TCP) > > (adresse IP / port TCP)
    - Deux versions : TCP et UDP

Principe du client-serveur

- Des processus veulent simultanément consulter et/ou modifier des données et des ressources se trouvant sur une machine central
- Ce qui implique des *conflits* et des problèmes d'intégrité des données (cf. BD)
- Remarque : Les problèmes peuvent être résolus en utilisant des verrous et/ou des sémaphores



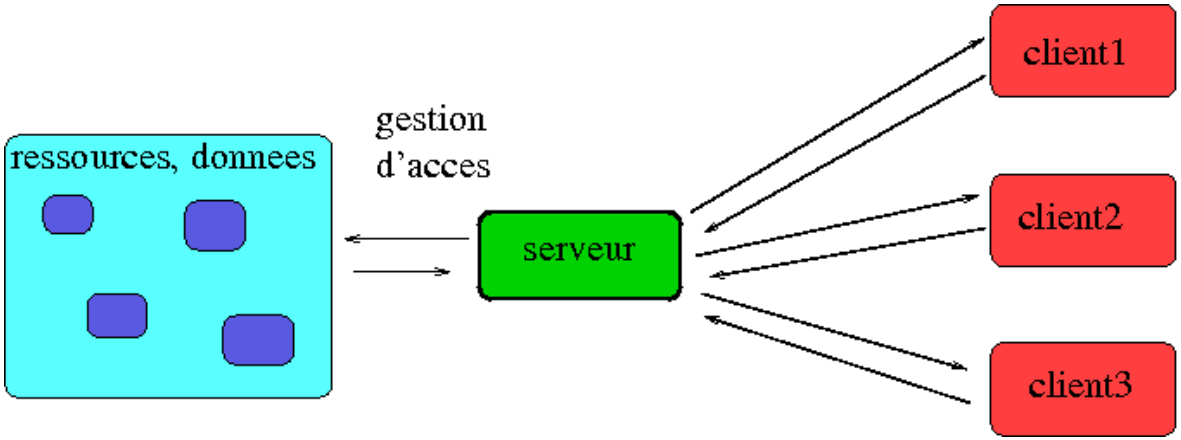
Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Principe du client-serveur

- Une autre possibilité plus systémique, robuste et facile à gérer est l'application du modèle client-serveur
  - n'autoriser qu'*un seul processus (serveur)* à consulter et modifier les données
  - chaque processus (client) désirant utiliser les données doit envoyer une requête au serveur
  - le serveur traite les requêtes et donne des réponses



- Séparation claire :
  - les clients ne s'occupent pas de l'organisation des données
  - le serveur ne s'occupe pas de l'utilisation des réponses

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Principe du client-serveur

- Exemples de serveurs
  - BD (Oracle, PostgreSQL, ...)
  - web (Apache, TOMCAT)
  - annuaire (LDAP)
  - mail (sortie : SMTP, entrée : POP3, IMAP)

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Principe du client-serveur

Les processus sur le serveur peuvent réaliser

- Serveur simple
  - Un client à la fois
- Serveur multi-processus
  - lors du traitement d'un service, il est possible qu'un même service soit demandé sans que le traitement du premier ne soit terminé
  - au moment d'une demande, le serveur crée une copie à l'aide de *fork()* (Le processus père reste libre ; si une autre demande arrive, un nouveau processus fils est créé)
  - lorsqu'une demande est satisfaite, le processus fils se termine et disparait
  - la même organisation peut être réalisée avec des pocessus léger (*threads*)

Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Principe du client-serveur

- La communication entre les clients et le serveur peut être organisée de plusieurs façons
  - Sur la même machine :
    - par tubes nommés
    - par files de messages, ...
  - Sur les machines différentes :
    - par sockets
  - Sur le serveur, entre les processus, on peut aussi gérer les messages par boîtes aux lettres

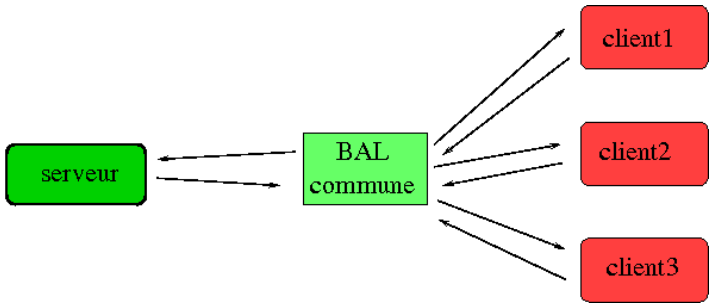
Miklós MOLNÁR

contacter
molnar@lirmm.fr

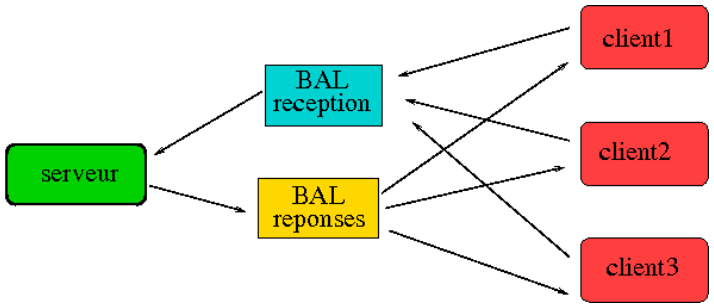
Principe du client-serveur

- Plusieurs organisations des boîtes aux lettres :

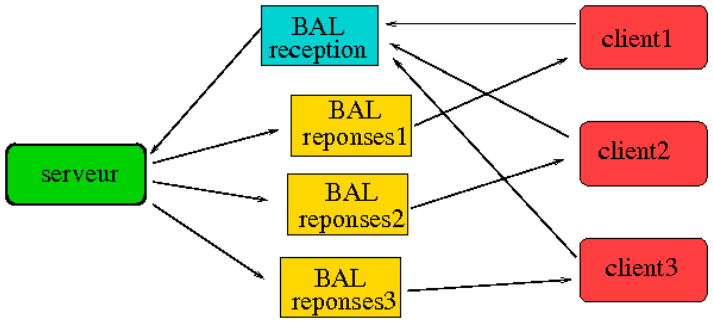
1) Une boîte pour tout le monde



2) Une boîte pour la réception des requêtes et une autre pour les réponses



3) Une boîte pour la réception des requêtes et une boîte par clients pour les réponses



Principe du client-serveur

Avantages

- Côté serveur :
  - le service tourne en permanence, attendant des requêtes
  - il peut répondre à plusieurs clients en même temps
  - machines serveur centralisées et peu nombreuses permettent de minimiser les redondances et des contradictions
  - meilleure sécurité : faible nombre de points d"entrée pour l'accès aux données
  - machines robustes et rapides, avec grande mémoire, disques suffisants, tolérantes aux pannes (?)

- Côté clients :
  - les clients ne s'occupe pas de l'implémentation
  - les clients ne sont pas des ressources critiques
  - facile à ajouter/enlever des clients sans perturber le fonctionnement

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Réseaux, modèle OSI

Miklós MOLNÁR

contacter
molnar@lirmm.fr

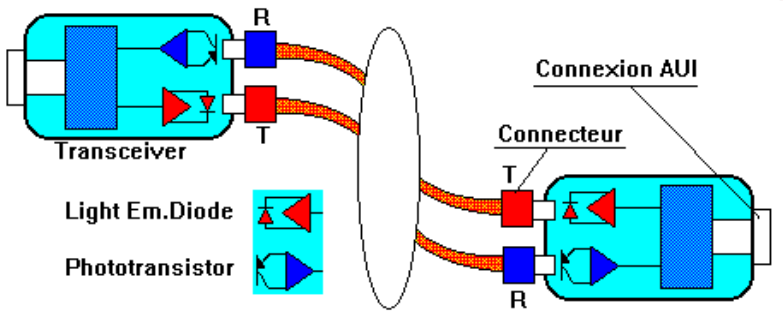
Découper les problèmes en couche a des avantages

- Chacune des couches représente une catégorie de problèmes, a ses propre fonctionnalités et offre des services transparents à la couche supérieure directe
- Chaque couche garantit à la couche supérieure que le service a été réalisé (sans erreur)
- Le découpage permet de changer la solution technique d'une couche sans changement pour les utilisateurs (couches supérieurs)

Identification des objets communicants (adressage)

Aux différents niveaux du réseau, différents objets assurent la communication et doivent être adressés
Comment les identifier ?

- Au niveau physique, les objets responsables sont directement concernés (liés)
  - un photo-diode est lié à un canon laser via une fibre optique
  - l'antenne d'une carte wifi est à la porté d'une station de base...



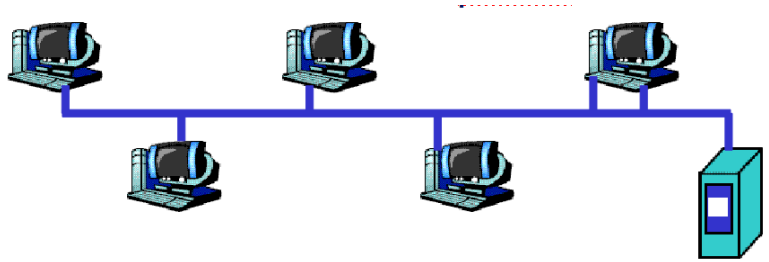
Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

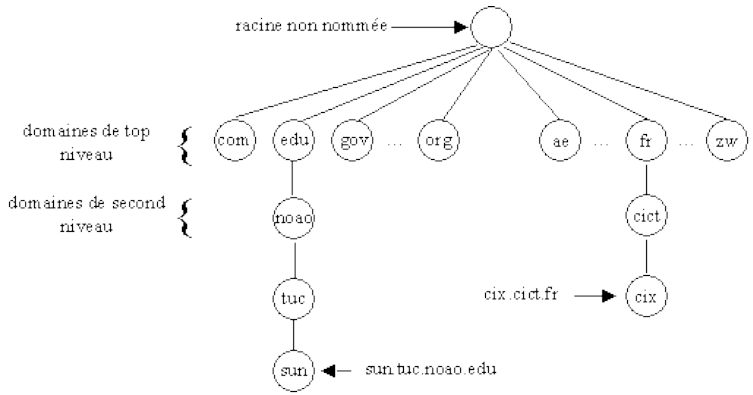
Identification des objets communicants (adressage)

- Au niveau d'un réseau local : "broadcast and select"
  - l'émetteur transmet la trame au réseau qui est alors entendue par "tout le monde"
  - le numéro de la carte réseau du destinataire indique la machine cible (et uniquement cette machine interprète la trame)
  - ce numéro (adresse MAC) est unique au monde
    - Format : 6 octets exprimés en hexadécimal, séparés par :
    - OUI (Organization Unique Id) (3 octets)
    - PID (Product Id) (3 octets)
    - Exemple : 01 :06 :4E :4B :14 :52
  - seule la carte destinataire selectionne le message, les autres l'ignore



Identification des objets communicants (adressage)

- Dans le réseau étendu, le routage basé sur des adresses IP des machines assure la transmission des messages
  - L'adresse IP est un identifiant fictif, donné par le réseau d'une façon stable ou dynamique
  - Le numérotation est hiérarchique avec une signification claire
  - Exemple : ens.univ-fc.fr
    le champ le plus à droite indique le nom de la zone, suivi par le nom du domaine et par le nom de la machine



Miklós MOLNÁR

contacter
molnar@lirmm.fr

Identification des objets communicants (adressage)

- Au niveau transport, c'est les services qui sont identifiés
  - Les applicatifs (des processus, des serveurs, des applications) sont très différents et tournent sous des systèmes différents. Identifier un applicatif sur une machine distante est difficile.
  - On utilise des *numéros* avec une technique de rendez-vous pour avoir le service
- Les ports TCP
  - Un port TCP/IP est un numéro de service et peut être vu comme un point de rendez-vous
  - Le programme serveur demande au système de *lui donner toutes les informations qui arrivent sur un port donné (canal d'écoute)*
  - Le programme client qui veut le service, doit utiliser le port spécifié sur la machine donnée (il doit connaître le numéro de port qui lui permettra de joindre le bon serveur)
- Etablissement d'une communication
  - Lorsque qu'un client veut communiquer avec un serveur, il demande à son système de lui donner un numéro de port libre (arbitraire)
  - Le port du serveur doit être connu (réservé)
  - Ainsi, les données partent d'un port d'une machine source vers un port sur une machine destination

Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Identification des objets communicants (adressage)

- Exemples de ports réservés

Port	Service ou Application
20,21	FTP
23	Telnet
25	SMTP
53	Domain Name System
63	Whois
70	Gopher
79	Finger
80	HTTP
110	POP3

voir la liste : http://www.emsisoft.net/fr/kb/portlist/

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Transmission des messages

- La couche transport peut fiabiliser la communication
- TCP
  - une connexion virtuelle est établie (trame TCP)
  - les paquets sont numérotés
  - la *destination contrôle* que tous les paquets sont arrivés et dans le bon ordre
  - si problème, retransmission des paquets
- UDP
  - pas de connexion virtuelle ni contrôle
  - transmission ”brute” selon l’adresse IP et le port

Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

URL (Uniform Ressource Locator ou adresse web)

- Il donne des informations sur l'accès à des informations/ressources
- Chaque document comporte sa propre adresse URL et est accessible par cette adresse à partir des navigateurs (document HTML, image, son, boîte aux lettres, etc.)
- Un URL est une *chaîne de caractères* comprenant
  - le protocole de communication, un nom d'utilisateur, un mot de passe, une adresse IP ou un nom de domaine, un numéro de port TCP/IP, un chemin d'accès, une requête

ftp ://Guest :guest01@ftp.ex.com :20/common/d/sources/cactus.h

http ://dictionnaire.tv5.org/dictionnaires.asp ?Action=1"mot=demarrer

  - Les URL peuvent être relatifs
    - Les URL relatives sont souvent utilisées pour les hyperliens à l'intérieur d'un même site web. Elles sont inspirées du système de fichiers Unix. Si le document d'URL http ://www.pb.fr/def référence l'URL relative forme1, cela correspond à http ://www.pb.fr/def/forme1

La classe URL en Java

- Elle permet de créer une URL et accéder à son contenu

```
final class URL {
 URL (String texte); ...
 URL (String protocol, String host, int port, String file)
             throws MalformedURLException;
 String  getFile();
 String  getHost();
 int   getPort();
 String  getProtocol();
...
 URLConnection openConnection () throws IOException;
 final InputStream openStream () throws IOException;
...
}
```

- Télécharger des données à partir des connexions :
  - par un objet `URLConnection` créé avec `openConnection()`
  - par un objet `InputStream` créé avec `openStream()`
  - ...

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemples

- A partir de URLConnection

```
public class ESSAI1 {
public static void main(String[] argv){
 try{
  URL url = new URL("http://www.lirmm.fr/xml/fr/lirmm.html");
  URLConnection connexion = url.openConnection();
  InputStream flux = connexion.getInputStream();
  }
  catch(MalformedURLException e){System.out.println(e);}
  catch(IOException e){System.out.println(e);}
  int len = connexion.getContentLength();
  ...
```

- A partir du flux de caractères

```
...
int len = url.getContentLength();
InputStream input = url.openStream(); // stream direct
...
for(;len != 0; len--)
 System.out((char)input.read());
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr

Manipulation des adresses IP depuis Java

- Utilisation d'un objet InetAdress pour connaître le nom ou l'adresse de l'ordinateur

```
final class InetAddress {
 static InetAddress getLocalHost();
 static InetAddress getByName(String host);
 byte[] getAddress();
 String getHostName();
}
```

  - pas de constructeur, mais des méthodes :

```
public static InetAddress getLocalHost();
//renvoie l'adresse internet de son ordinateur
public static InetAddress getByName(String host);
//renvoie l'adresse de l'ordinateur passé en paramètre
//"host" peut etre un numéro IP ou le nom de l'ordinateur
byte[] getAddress();
//renvoie l'adresse IP dans un tableau d'octets
String getHostName();
//renvoie le nom de l'ordinateur
String getHostAddress()
//renvoie l'adresse IP
```

Exemple de l'utilisation de InetAddress

```
 try {
      InetAddress adrLoc = InetAddress.getLocalHost();
      InetAddress adrRem = InetAddress.getByName("java.sun.com");
      ...
      System.out.println(adrLoc.getHostAddress());
           // adresse IP de la machine locale...
} catch (Exception e) { e.printStackTrace();
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr

Paquets, datagrammes

Classe DatagramPacket

Objets qui contiennent les données envoyées ou reçues ainsi que l'adresse de destination et de source du datagramme

- Pour envoyer les messages (avec IP ou en mode UDP)

  public DatagramPacket(byte buffer[], int taille, InetAddress adr, int port)

  - buffer est un tableau d'octets utilisé pour la communication
  - taille indique le nombre d'octets du message
  - adr est l'adresse du destinataire
  - port est le no de communication utilisé pour l'échange

- Pour recevoir un datagramme

  public DatagramPacket(byte buffer[], int taille)

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Paquets, datagrammes

Classe DatagramPacket

Quelques méthodes

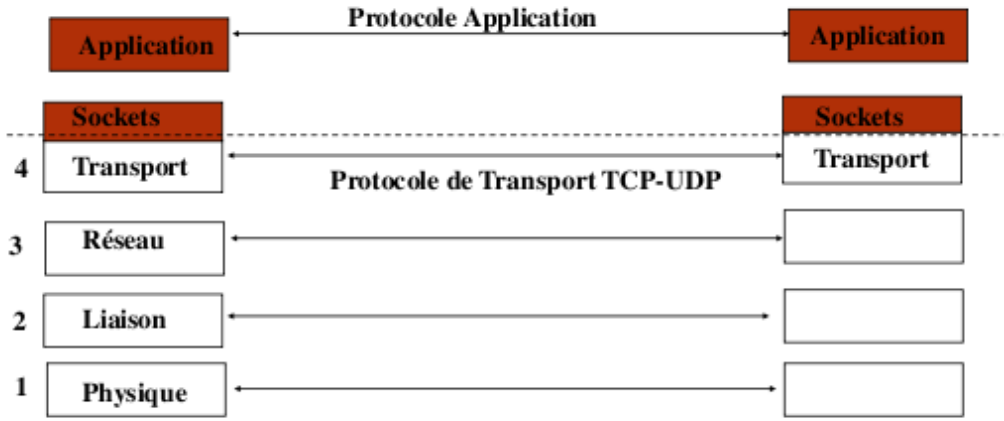
- public synchronized InetAddress getAddress ()
  Retourne l'adresse stockée dans le paquet
- public synchronized int getPort ()
  Retourne le port stocké dans le paquet
- public synchronized byte[] getData ()
  Retourne les données stockées dans le paquet
- public synchronized int getLength ()
  Retourne la taille des données stockées dans le paquet
- public synchronized void setAddress(InetAddress iaddr)
  Modifie ou affecte l'adresse de destination
- public synchronized void setPort(int iport)
  Modifie ou affecte le port de destination
- public synchronized void setData(byte ibuf[])
  Modifie ou affecte la référence de la zone contenant les données

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Sockets

- Modèle simple permettant la communication inter-processus à travers un réseau TCP/IP
- Il s'agit d'un point d'accès (un descripteur) aux services de la couche transport, c'est-à-dire de TCP ou UDP



- La communication par sockets adopte un modèle client-serveur
  - pour communiquer il faut créer un serveur prêt à recevoir les requêtes et des clients

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Sockets

Miklós MOLNÁR

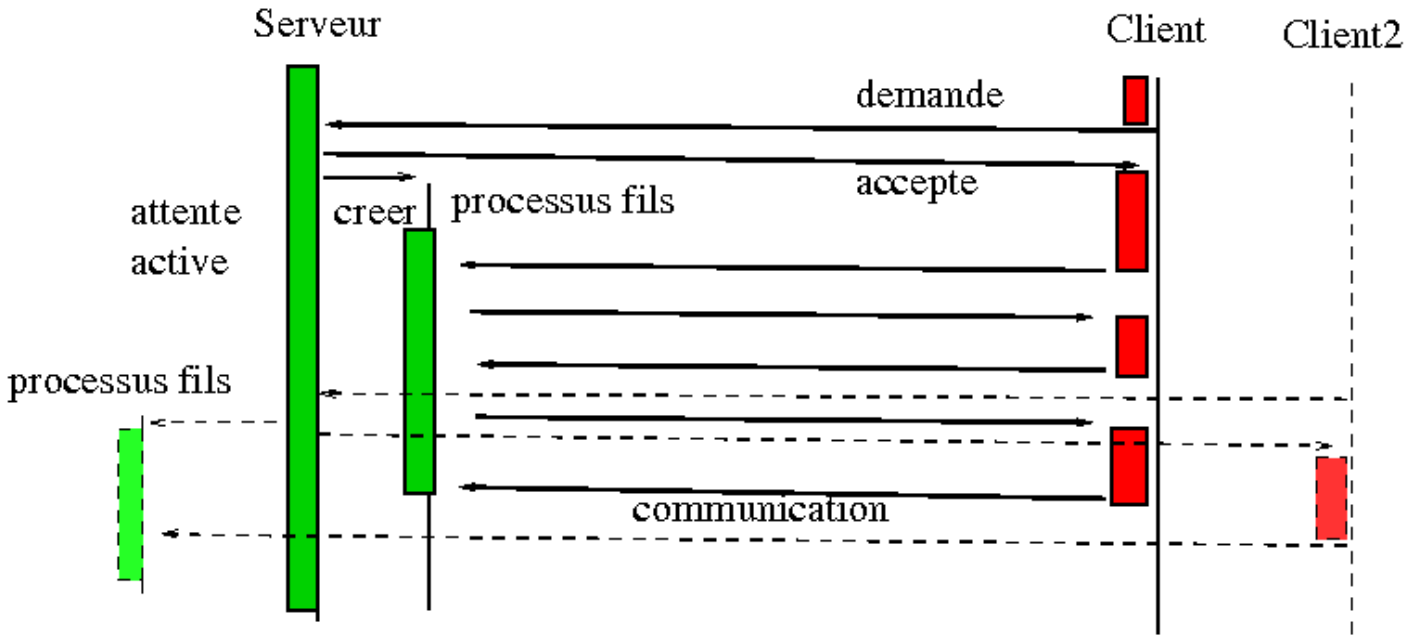
contacter
molnar@lirmm.fr

Deux modes de communication :

- Le mode non connecté utilisant le protocole UDP
  - il nécessite l'adresse de destination à chaque envoi
  - aucun accusé de réception n'est donné
- Le mode connecté utilisant le protocole TCP
  - une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données

Client-serveur avec des sockets

- Le *serveur* doit écouter les requêtes
- Eventuellement, exécution concurrente de plusieurs requêtes
  - Plusieurs processus (une mémoire associée à chaque processus)
  - Plusieurs processus légers (thread) dans le même espace virtuel (contexte restreint : pile, mot d'état, registres)
- Le *client* envoie sa demande du service (il est en général suspendu lors de l'exécution de la requête)



Miklós MOLNÁR

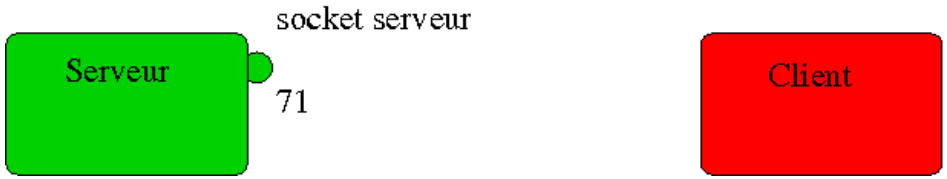
contacter
molnar@lirmm.fr

Client-serveur avec des sockets

Principe de fonctionnement : 3 phases

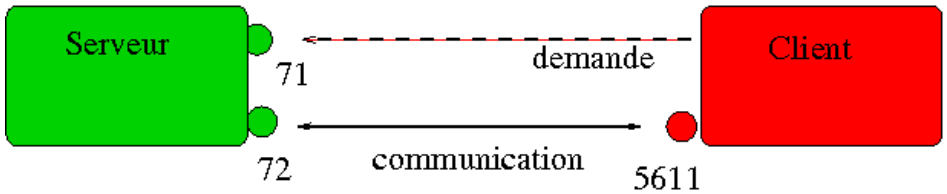
- Supposons TCP

1. Avant tout, le serveur crée une ”socket serveur” (associée à un port) et se met en attente



2. Le client se connecte à la ”socket serveur” et demande une connexion : deux sockets sont alors crées
   - une ”socket client” côté client
   - une ”socket service client” côté serveur

   Ces sockets sont connectées entre elles



3. Le client et le serveur communiquent par les sockets. Les méthodes sont celles des fichiers (read, write)

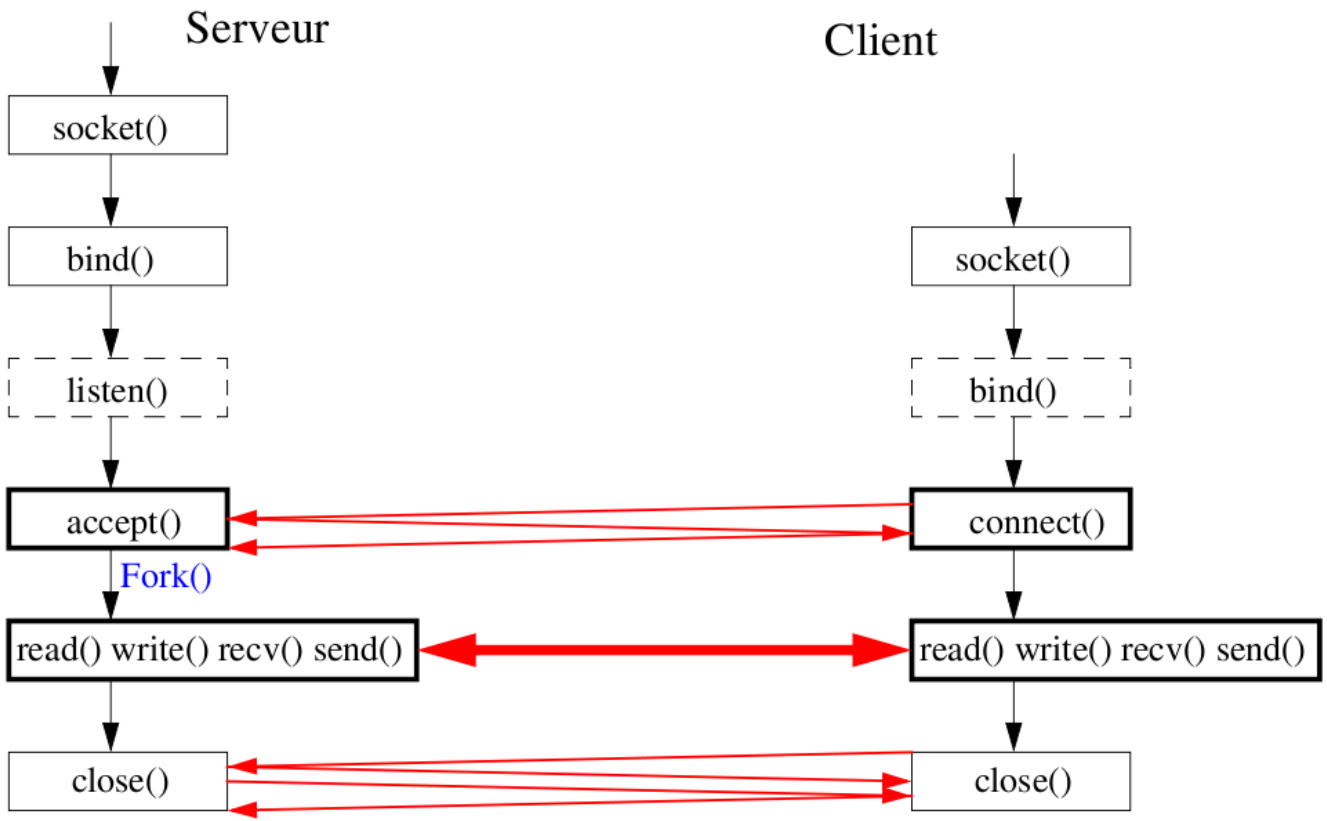
Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Client-serveur en mode connecté (TCP)

Le schéma vu en C



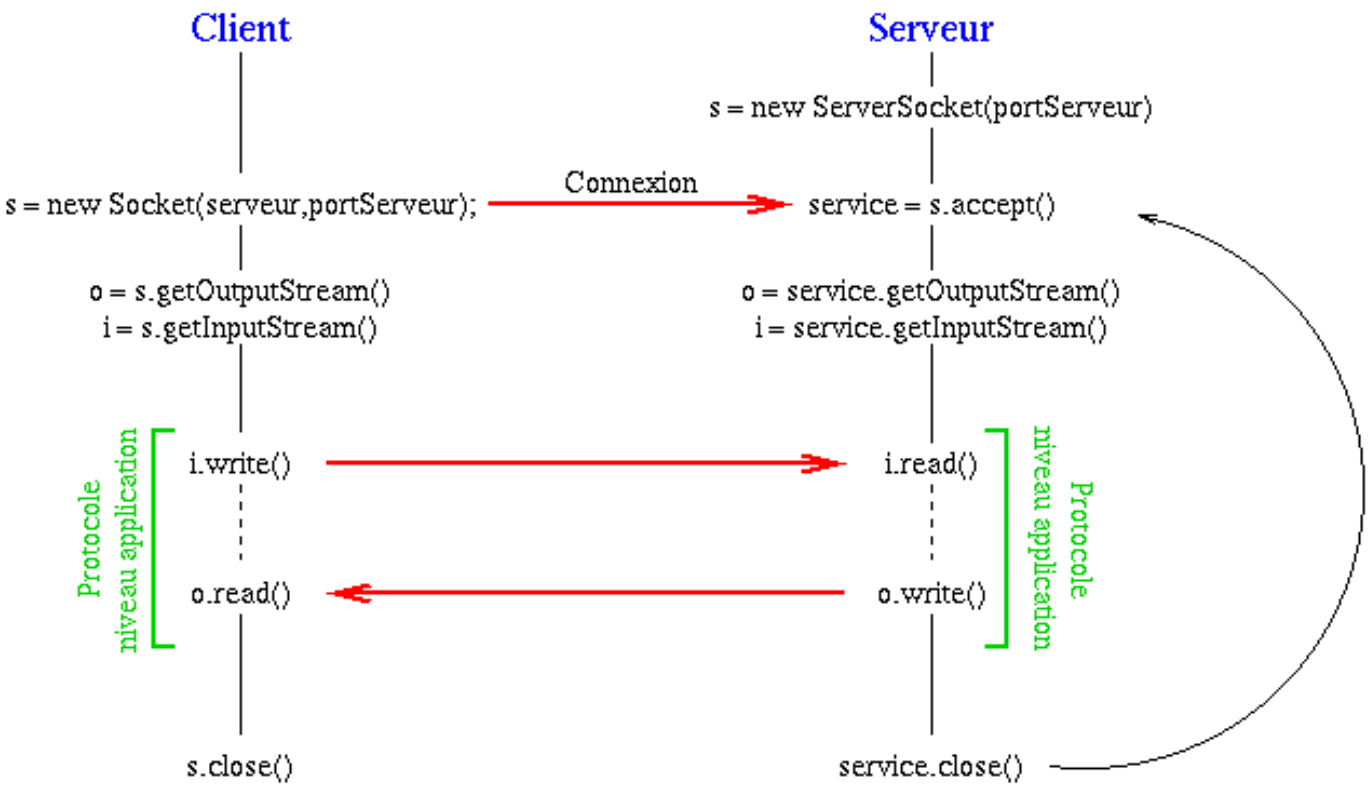
Chaque “close()” ne ferme qu’un seul sens de communication !

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Client-serveur en Java et en mode connecté (TCP)

Le schéma en Java



Miklós MOLNÁR

contacter
molnar@lirmm.fr

Serveur pour écouter le port

La classe ServerSocket

Elle propose un objet serveur via une interface socket
public class java.net.ServerSocket

Pour créer une socket serveur à l'écoute d'un port spécifié :

- public ServerSocket(int port) throws IOException
- public ServerSocket(int port, int backlog) throws IOException
- public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException
  - La taille de la file d'attente des demandes de connexion peut être explicitement spécifiée via le paramètre backlog
  - Si la machine possède plusieurs adresses, on peut aussi restreindre l'adresse sur laquelle on accepte les connexions
  - Remarque : ils correspondent à l'utilisation des primitives socket(), bind() et listen()

Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Serveur pour écouter le port

La classe ServerSocket

- Méthodes pour manipuler les sockets :
  - public Socket accept() throws IOException
    La méthode correspondante à l'acceptation d'une connexion d'un client. Cette méthode est bloquante
  - public void setSoTimeout(int timeout) throws SocketException
    Cette méthode détermine le délai de garde exprimé en millisecondes. La valeur par défaut 0 équivaut à l'infini
  - public synchronized int getSoTimeout() throws SocketException
    Retourne la valeur courante du timeout associé à la socket

- Autres méthodes :
  - public void close ()
    Ferme la socket et libère les ressources
  - public int getLocalPort ()
    Retourne le port d'attachement de la socket
  - public InetAddress getInetAddress()
    Retourne l'adresse IP de la socket

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple du fonctionnement d'une tâche socket serveur

```
import java.io.*;
import java.net.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class SoServeur implements Runnable [
        ExecutorService es; /** le groupe de tâches */
        ServerSocket sockserv=null; /** socket  serveur pour attendre les  clients */
        ServeurCli  commcli = null; /**  objet de communication avec un client */

        public SoServeur( ExecutorService es)
        {       try
                {       this.es = es;
                        sockserv = new ServerSocket (Port);    //création socket serveur
                }       catch (IOException ex) {  }
        }
        /** boucle d'attente de requêtes en provenance d'un client
         *  création d'un objet Socket pour un client */
        public void run()
        {       try
                {       while (true)
                        {       try
                                {       Socket sockcli = sockserv.accept(); //attente requête
                                        commcli= new ServeurCli (es, sockcli);
                                        es.execute(commcli);        //création tâche
                                }   catch (IOException ex) {  }
                        }
                }
                finally
                {       try
                        {       sockserv.close(); // fermeture port de communication
                                es.shutdown();
                        } catch (IOException ex) {      }
                }
        }

        public static void main (String args[]) throws Exception
                ExecutorService es = Executors.newFixedThreadPool(4);
            SoServeur serv = new SoServeur( es);
                es.execute(serv);
        }
}
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr

Client-serveur en mode connecté (Java)

La classe Socket

Cette classe est utilisée pour la programmation des sockets connectés, côté client et côté serveur

public class java.net.Socket

- Pour créer une socket côté serveur :
  la méthode `accept()` de la classe `ServerSocket`
  renvoie une socket de service connectée au client
- Côté client, on a des constructeurs :
  - public Socket(String host, int port) throws UnknownHostException, IOException
  - public Socket(InetAddress address, int port) throws IOException
  - public Socket(String host, int port, InetAddress localAddr, int localPort throws UnknownHostException, IOException
  - public Socket(InetAddress addr, int port, InetAddress localAddr, int localPort throws IOExceptionn
- Les deux premiers constructeurs créent une socket connectée à la machine et au port spécifiés
- Les deux suivants permettent aussi de fixer l'adresse IP et le numéro de port du client
- Par défaut, la connexion est de type TCP (fiable)

Client-serveur en mode connecté (Java)

La classe Socket
Communication via la classe Socket

- On utilise des flots de données (java.io.OutputStream et java.io.InputStream)
- Pour obtenir les flots en entrée et en sortie :
  - public InputStream getInputStream() throws IOException
  - public OutputStream getOutputStream() throws IOException
- Les flots permettent d'utiliser read() et write(), etc.
- Ils permettent la construction d'objets plus abstraites telles que java.io.DataOutputStream et java.io.DataInputStream (pour JDK1), ou java.io.PrintWriter et java.io.BufferedReader (pour JDK2)
- La lecture est bloquante tant que des données ne sont pas disponibles
- Il est possible de fixer une délai de garde à l'attente de données

  public void setSoTimeout(int timeout) throws SocketException

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Client-serveur en mode connecté (Java)

La classe Socket

Quelques méthodes

- public InetAddress getInetAddress() pour obtenir l'adresse IP distante
- public InetAddress getLocalAddress() pour obtenir l'adresse IP locale
- public int getPort() pour obtenir le port distant
- public int getLocalPort() pour obtenir le port local
- public void close() pour fermer la connexion et libèrer les ressources

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple de la communication via TCP

- Deux processus veulent échanger des messages (des lignes de texte)
  - d'abord, le serveur doit être créé
  - le client se connecte au serveur et commence l'émission des messages
  - le serveur lui répond par un écho de la ligne reçue
  - au bout de 10 échanges, le client envoie une message "END" et ferme la connexion (le serveur doit fermer sa connexion en recevant "END")

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple de la communication via TCP

- Le serveur

```
import java.io.*;
import java.net.*;
public class Serveur {
   static final int port = 8080;
   public static void main(String[] args) throws Exception {
        ServerSocket s = new ServerSocket(port);
        Socket soc = s.accept();
                // Un BufferedReader permet de lire par ligne
        BufferedReader ins = new BufferedReader(
                              new InputStreamReader(soc.getInputStream())
                             );

        // Un PrintWriter possède toutes les opérations print classiques.
        // En mode auto-flush, le tampon est vidé (flush) à l'appel de println.
        PrintWriter outs = new PrintWriter( new BufferedWriter(
                            new OutputStreamWriter(soc.getOutputStream())), true);
        while (true) {
           String str = ins.readLine();          // lecture du message
           if (str.equals("END")) break;
           System.out.println("ECHO = " + str);   // trace locale
           outs.println(str);                     // renvoi d'un écho
        }
        ins.close();
        outs.close();
        soc.close();
   }
}
```

Exemple de la communication via TCP

- Le client

```
import java.io.*;
import java.net.*;
/** Le serveur est fourni dans la commande    */
public class Client {
   static final int port = 8080;

   public static void main(String[] args) throws Exception {
        Socket socket = new Socket(args[0], port);
        System.out.println("SOCKET = " + socket);
        BufferedReader ins = new BufferedReader(
                              new InputStreamReader(socket.getInputStream()) );
        PrintWriter outs = new PrintWriter(  new BufferedWriter(
                             new OutputStreamWriter(socket.getOutputStream())), true);

        String str = "bonjour";
        for (int i = 0; i < 10; i++) {
           str = str + Integer.toString(i);
           outs.println(str);          // envoi d'un message
           str = ins.readLine();      // lecture de l'écho
        }
        System.out.println("END");     // message de terminaison
        outs.println("END") ;
        ins.close();
        outs.close();
        socket.close();
   }
}
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple de la communication via TCP

Miklós MOLNÁR

contacter
molnar@lirmm.fr

- Traces d'exécution
  - sur le serveur

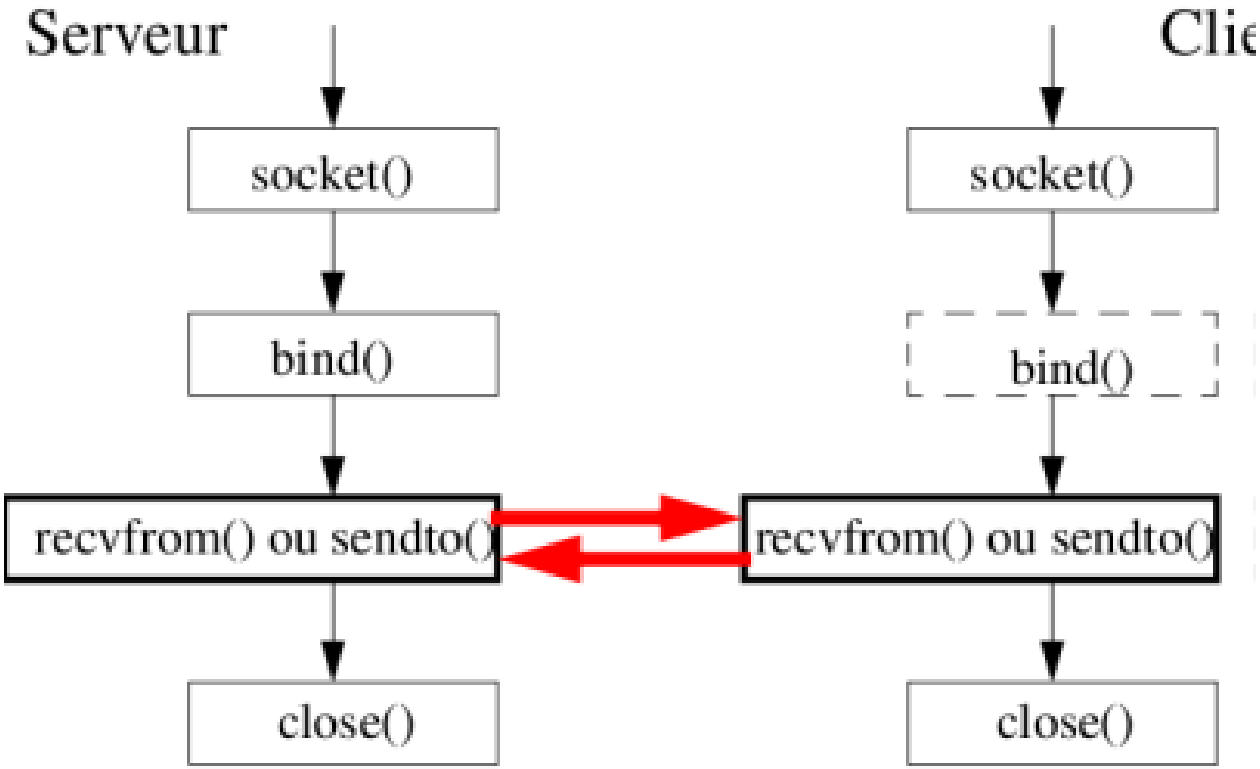
```
miklos@miklos-laptop $ java Serveur
ECHO = bonjour0
ECHO = bonjour01
ECHO = bonjour012
ECHO = bonjour0123
ECHO = bonjour01234
ECHO = bonjour012345
ECHO = bonjour0123456
ECHO = bonjour01234567
ECHO = bonjour012345678
ECHO = bonjour0123456789
miklos@miklos-laptop $
```

  - sur le client

```
miklos@miklos-laptop $ java  Client miklos-laptop
SOCKET = Socket[addr=miklos-laptop/127.0.1.1,port=8080,localport=40975]
END
miklos@miklos-lapto $
```

Client-serveur en mode non-connecté (UDP)

Le schéma du système



Miklós MOLNÁR

contacter
molnar@lirmm.fr

Client-serveur en mode non-connecté (Java)

Miklós MOLNÁR

contacter
molnar@lirmm.fr

La classe DatagramSocket

- Pour créer une socket du coté client et du coté serveur, les constructeurs :
  - public DatagramSocket () throws SocketException
    crée la socket et l'attache à un port disponible de la machine locale
    »» Il est utilisé dans les clients pour lesquels le port d'attachement n'a pas besoin d'être défini
  - public DatagramSocket (int port) throws SocketException
    Crée la socket et l'attache au port UDP local passé en paramètre
    »» Dans les serveurs pour lesquels le port d'attachement a besoin d'être fixé préalablement afin qu'il soit connu des clients

Client-serveur en mode non-connecté (Java)

La classe DatagramSocket

- On peut envoyer et recevoir des datagrammes, via la socket, à l'aide des méthodes suivantes :
  - public void send(DatagramPacket data) throws IOException
    Permet d'envoyer les données contenues dans data vers la machine et le port préalablement spécifiés (dans la variable data)
  - public synchronized void receive(DatagramPacket data) throws IOException
    Permet de recevoir un datagramme qui sera stocké dans data. Après appel, data contient les données reçues, leur taille, l'adresse de l'envoyeur ainsi que son port d'attachement.
    Cette méthode est bloquante tant qu'il n'y a rien à recevoir.
    Si le message est trop long pour être stocké, celui-ci est tronqué, et le reste est perdu. Il n'est donc pas possible de recevoir des messages dont on ne connait pas préalablement la taille.
- On peut spécifier un délai d'attente maximal en reception :
  public synchronized void setSoTimeout(int timeout) throws SocketException

Miklós MOLNÁR

contacter
molnar@lirmm.fr

I.U.T MONTPELLIER

Client-serveur en mode non-connecté (Java)

Miklós MOLNÁR

contacter
molnar@lirmm.fr

La classe DatagramSocket

- D'autres méthodes pour manipuler les sockets :
  - public void close ()
    ferme la socket et libère les ressources
  - public int getLocalPort ()
    retourne le port d'attachement de la socket
  - public synchronized int getSoTimeout() throws SocketException
    retourne la valeur courante du timeout associé à la socket

Quasi-connexion en mode UDP

- Il est possible de ”connecter” une socket datagramme à un destinataire
  - les paquets émis sur la socket seront toujours pour l'adresse spécifiée
  - la connexion simplifie l'envoi d'une série de paquets (il n'est plus nécessaire de spécifier l'adresse de destination pour chaque paquet)
- la ” déconnexion ” supprime l'association (la socket redevient dans l'état initial)

- public void connect(InetAddress address, int port)
- public void disconnect()

Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple pour client-serveur en mode UDP

Dans cette application, le client envoie des messages numérotés,

- La classe client

```
import java.io.*;
import java.net.*;
public class ClientUDP {
 final static int taille = 10;
 final static byte buffer[] = new byte[taille];

 public static void main(String argv[]) throws Exception {
 InetAddress serveur = InetAddress.getByName(argv[0]);
   // nom du serveur passe en parametre
 System.out.println("nom: " + serveur.getHostName());

 DatagramSocket socket = new DatagramSocket();
 DatagramPacket dataRecieved = new DatagramPacket(new byte[taille],taille);
for(int ii = 1; ii < 10; ii++) {
  String s = Integer.toString(ii);
   byte buffer[] = s.getBytes();
   int l = s.length();
   DatagramPacket dataSent = new DatagramPacket(buffer,l,serveur,ServeurUDP.port);
   System.out.println("Envoie : " + new String(dataSent.getData()));
   socket.send(dataSent);
   socket.receive(dataRecieved);
   System.out.println("Reponse : " + new String(dataRecieved.getData()));
   System.out.println("De : " + dataRecieved.getAddress() + ":" + dataRecieved.getPort());
  }
 }
}
```

Exemple pour client-serveur en mode UDP

Dans cette application, le serveur renvoie des ACKs (ici, les numéros)

- la classe serveur

```
import java.io.*;
import java.net.*;

class ServeurUDP {
 final static int port = 8532;
 final static int taille = 10;
 static byte tampon[] = new byte[taille];

 public static void main(String argv[]) throws Exception {
 DatagramSocket socket = new DatagramSocket(port);
 DatagramPacket data = new DatagramPacket(tampon,tampon.length);
 while(true) {
    socket.receive(data);
    System.out.println(new String(data.getData()));
    System.out.println(data.getAddress());
    socket.send(data);
  }
 }
}
```

Exemple pour client-serveur en mode UDP

- l'exécution du client

```
miklos@miklos-laptop $ java ClientUDP miklos-laptop
nom: miklos-laptop
Envoie : 1
Reponse : 1
De : /127.0.1.1:8532
Envoie : 2
Reponse : 2
De : /127.0.1.1:8532
Envoie : 3
Reponse : 3
De : /127.0.1.1:8532
Envoie : 4
Reponse : 4
De : /127.0.1.1:8532
Envoie : 5
Reponse : 5
De : /127.0.1.1:8532
Envoie : 6
Reponse : 6
De : /127.0.1.1:8532
Envoie : 7
Reponse : 7
De : /127.0.1.1:8532
Envoie : 8
Reponse : 8
De : /127.0.1.1:8532
Envoie : 9
Reponse : 9
De : /127.0.1.1:8532
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr

Exemple pour client-serveur en mode UDP

- l'exécution du serveur

```
miklos@miklos-laptop $ java ServeurUDP
1
/127.0.1.1
2
/127.0.1.1
3
/127.0.1.1
4
/127.0.1.1
5
/127.0.1.1
6
/127.0.1.1
7
/127.0.1.1
8
/127.0.1.1
9
/127.0.1.1
```


Miklós MOLNÁR

contacter
molnar@lirmm.fr