Les sockets

Daniel Hagimont

IRIT/ENSEEIHT
2 rue Charles Camichel - BP 7122
31071 TOULOUSE CEDEX 7

Daniel.Hagimont@enseeiht.fr
http://hagimont.perso.enseeiht.fr

Préambule
Modèle client-serveur

- Application client/serveur
  - application qui fait appel à des services distants au travers d'un échange de messages (les requêtes) plutôt que par un partage de données (mémoire ou fichiers)
  - serveur
    - programme offrant un service sur un réseau (par extension, machine offrant un service)
  - client
    - programme qui émet des requêtes (ou demandes de service). Il est toujours l'initiateur du dialogue

Préambule
Exemple de client/serveur

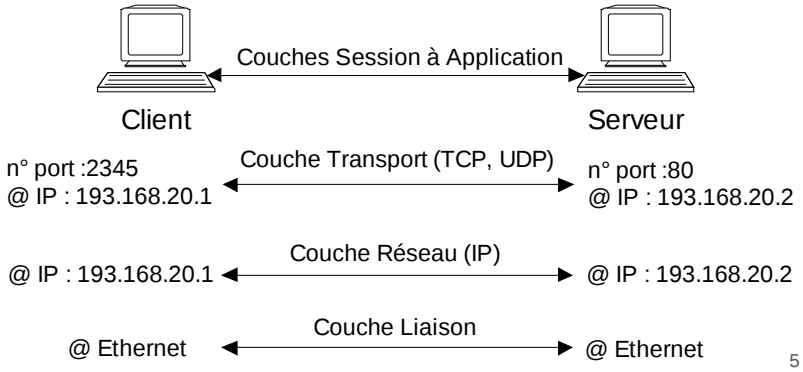
- Un serveur de fichiers
- Des clients qui demandent des fichiers
- Comment gérer la concurrence ?
  - un processus serveur ⇒ un seul client servi à la fois
  - plusieurs clients ⇒ il va falloir faire des fork() !!!
- Quel protocole utiliser ?
  - le client envoie le nom du fichier
  - le serveur renvoie la taille puis les données
  - comment gère-t-on les erreurs ?

Mode
connecté/non connecté

- Mode connecté (TCP)
  - problèmes de communications gérés automatiquement
  - primitives simples d'émission et de réception
  - gestion de la connexion coûteuse
  - pas de délimitation des messages dans le tampon
- Mode non connecté (UDP)
  - consomme moins de ressources systèmes
  - permet la diffusion/multicast
  - gestion de toutes les erreurs à la main : il faut réécrire la couche transport !!!

Les sockets

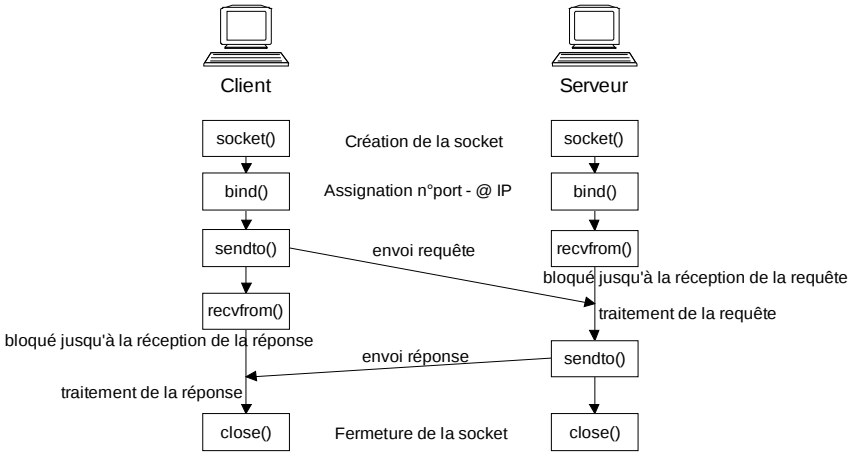
- Interface d'accès au réseau
- développé dans Unix BSD
- n° port, @ IP, protocole (TCP, UDP, ...)



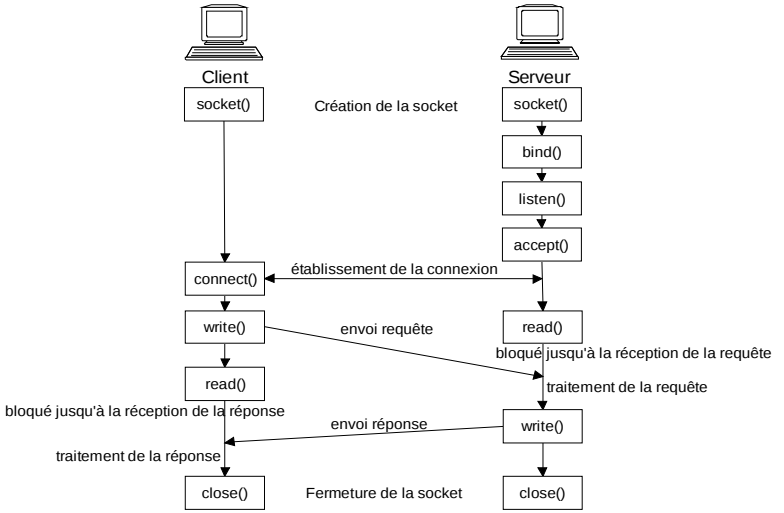
L'API socket

- Création de socket : socket(family, type, protocol)
- Ouverture de dialogue :
  - client : bind(..), connect(...)
  - serveur : bind(..), listen(...), accept(...)
- Transfert de données :
  - mode connecté : read(...), write(...), send(...), recv(...)
  - mode non connecté : sendto(...), recvfrom(...), sendmsg(...), recvmsg(...)
- Clôture du dialogue :
  - close(...), shutdown(...)

Client/Serveur en mode non connecté



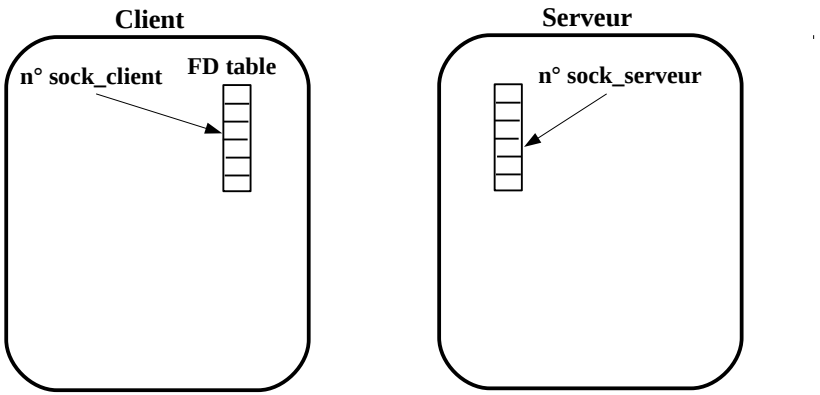
Client/Serveur en mode connecté



La primitive socket()

- int socket(int family, int type, int protocol)
- family
  - AF_INET : pour des communications Internet
  - AF_UNIX : pour des communications locales
- type ou mode de fonctionnement
  - SOCK_STREAM : mode connecté (TCP)
  - SOCK_DGRAM : mode déconnecté (UDP)
  - SOCK_RAW : accès direct aux couches basses (IP)
- protocol :
  - Protocole à utiliser, normalement il n'y en a qu'un
  - 0 par défaut

Après appel à socket()

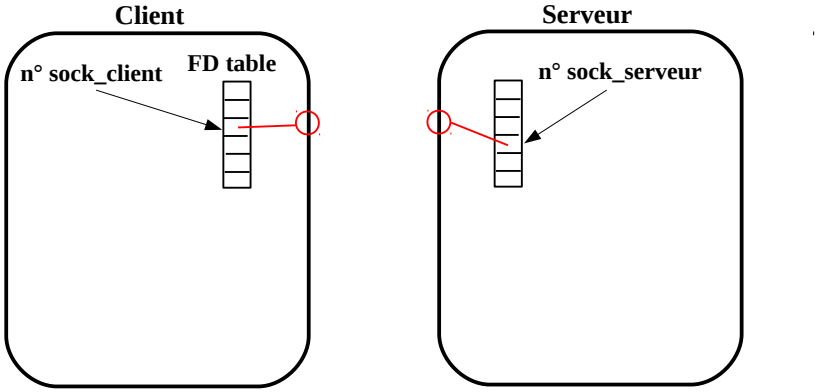


La primitive bind()

- **int bind(int sock_desc, struct sockaddr *my_@, int lg_@)**
- **sock_desc : descripteur de socket retourné par socket()**
- **my_@ : adresse IP et n° de port auxquels le serveur veut répondre**
- **Exemple de serveur :**

```
int sd;
struct sockaddr_in server; // @IP, n° port, mode

sd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = 0; // Laisse le système choisir un port
server.sin_addr.s_addr = INADDR_ANY;
                        // n'importe quelle interface réseau

bind(sd, (struct sockaddr *)&server, sizeof(server));
```


Après appel à bind()



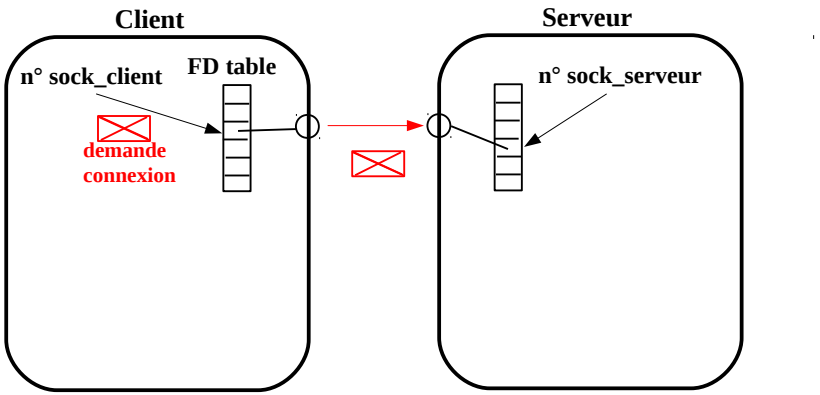
On peut déjà s'échanger des messages si on est en mode non connecté

La primitive connect()

- **int connect(int sock_desc, struct sockaddr * @_server, int lg_@)**
- **sock_desc : descripteur de socket retourné par socket()**
- **@_serveur : adresse IP et n° de port du serveur distant**
- **Exemple de client :**

```
int sd;
struct sockaddr_in server; // @IP, n° port, mode
struct hostent     remote_host; // nom et @IP

sd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(13);
remote_host = gethostbyname("brassens.upmf-grenoble.fr");
bcopy(remote_host->h_addr, (char *)&server.sin_addr,
       remote_host->hlength); // Recopie de l'adresse
connect(sd, (struct sockaddr *)&server, sizeof(server));
```


Après appel à connect()



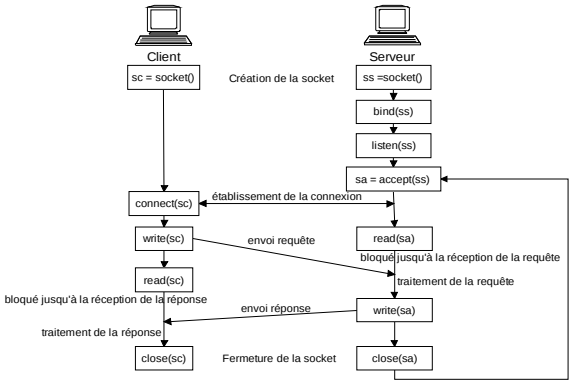
La primitive listen()

- **int listen(int sock_desc, int nbr)**
- **sock_desc : descripteur de socket retourné par socket()**
- **nbr : nombre maximum de connexions en attente d'être acceptées**
- **Exemple de serveur :**

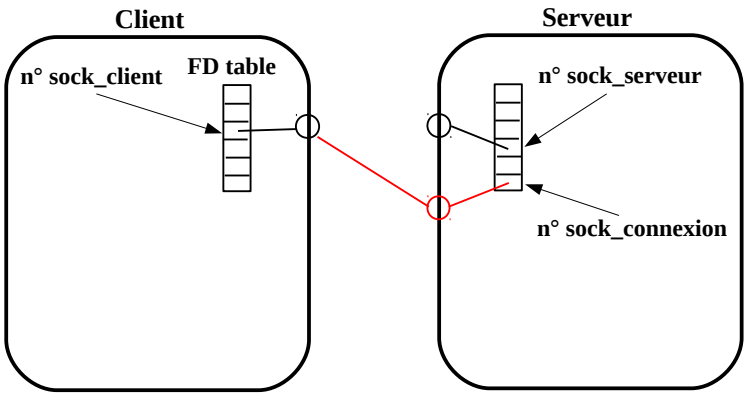
```
int sd;
struct sockaddr_in server; // @IP, n° port, mode

sd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = 0; // Laisse le système choisir un port
server.sin_addr.s_addr = INADDR_ANY;
                    // n'importe quelle interface réseau
bind(sd, (struct sockaddr *)&server, sizeof(server));
listen(sd, 5);
```


La primitive accept()

- int accept(int sock_desc, struct sockaddr *client, int lg_@)
- sock_desc : descripteur de socket retourné par socket()
- client : identité du client demandant la connexion
- accept renvoie le descripteur de la nouvelle socket créée



Après appel à accept()

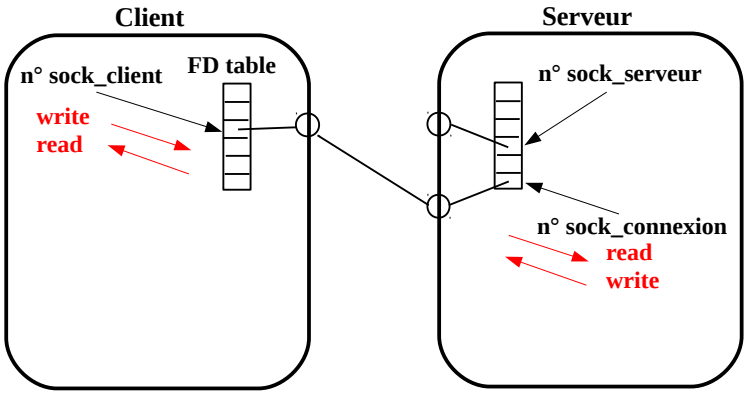


Les primitives d'envoi/réception

- `int write(int sock_desc, char *buff, int lg_buff);`
- `int read(int sock_desc, char *buff, int lg_buff);`
- `int send(int sock_desc, char *buff, int lg_buff, int flag);`
- `int recv(int sock_desc, char *buff, int lg_buff, int flag);`
- `int sendto(int sock_desc, char *buff, int lg_buff, int flag, struct sockaddr *to, int lg_to);`
- `int recvfrom(int sock_desc, char *buff, int lg_buff, int flag, struct sockaddr *from, int lg_from);`

- `flag` : options de contrôle de la transmission (consulter le man)

Communication



Un serveur concurrent sous Unix

- Lors du fork() le fils hérite des descripteurs du père
- Exemple de serveur :

```
int sd, nsd;
...
sd = socket(AF_INET, SOCK_STREAM, 0);
...
bind(sd, (struct sockaddr *)&serveur, sizeof(serveur));
listen(sd, 5);
while (!fin) {
     nsd = accept(sd, ...);
     if (fork() == 0) {
             close(sd); // On n'a plus besoin de la socket du père

             /* On traite ici la connexion avec le client */

             close(nsd); // Fin de la connexion avec le client
             exit(0); // Mort du fils
     }
     close(nsd); // Le père n'a plus besoin de la socket vers le client
}
```

Programmation Socket en Java

- package **java.net**
  - **InetAddress**
  - **Socket**
  - **ServerSocket**
  - **DatagramSocket / DatagramPacket**

Utilisation de **InetAddress** (1)

```
public class Inria {

  public static void main (String[] args) {
    try {
      InetAddress adresse =
        InetAddress.getByName("www.inria.fr");
      System.out.println(adresse);
    } catch (UnknownHostException e) {
      System.out.println
        ("Impossible de trouver www.inria.fr");
    }
  }
}
```


Utilisation de **InetAddress** (2)

```
public class ExINT {

  public static void main (String[] args) {
    try {
      InetAddress adresse =
        InetAddress.getByName("157.159.110");
      System.out.println(adresse);
    } catch (UnknownHostException e) {
      System.out.println
        ("Impossible de trouver 157.159.110");
    }
  }
}
```


Accès aux valeurs

```
public class QuiSuisJe {

  public static void main (String[] args) {
    try {
      InetAddress a = InetAddress.getLocalHost();
      System.out.println(a.getHostName() + " / " +
                         a.getHostAdress());
    } catch (UnknownHostException e) {
      System.out.println
        ("Pas d'accès à mon adresse");
    }
  }
}
```

Socket client et connexion TCP

```
try {
    Socket s = new Socket("www.inria.fr",80);
  …
} catch (UnknownHostException u) {
    System.out.println("hôte inconnu");
} catch (IOException e) {
    System.out.println("IO exception");
}
```


Lecture/écriture TCP

```
try {
    Socket s = new Socket ("www.inria.fr",80);
    InputStream is = s.getInputStream();
    …
    OutputStream os = s.getOutputStream();
    …
} catch (Exception e) {
    System.err.println(e);
}
```


Socket serveur TCP

```
try {
    ServerSocket serveur = new ServerSocket(port);
    Socket s = serveur.accept();
    OutputStream os = s.getOutputStream();
    InputStream is = s.getIntputStream();
    …
} catch (IOException e) {
    System.err.println(e);
}
```


Quelques mots sur les classes de gestion des flux

- Suffixe
  - Flux d'octet (InputStream/OutputStream)
  - Flux de caractères (Reader/Writer)
- Préfixes
  - Flux (source ou destination)
    - ByteArray, File, Object ...
  - Filtre (type de traitement)
    - Buffered, LineNumber, ...

Quelques mots sur les classes de gestion des flux

	Flux en lecture	Flux en sortie
Flux de caractères	BufferedReader CharArrayReader FileReader InputStreamReader LineNumberReader PipedReader PushbackReader StringReader	BufferedWriter CharArrayWriter FileWriter OutputStreamWriter PipedWriter StringWriter
Flux d'octets	BufferedInputStream ByteArrayInputStream DataInputStream FileInputStream ObjectInputStream PipedInputStream PushbackInputStream SequenceInputStream	BufferedOutputStream ByteArrayOutputStream DataOuputStream FileOutputStream ObjetOutputStream PipedOutputStream PrintStream

Quelques mots sur les classes de gestion des flux

```
BufferedReader br = new BufferedReader(
   new InputStreamReader(socket.getInputStream()));
String s = br.readLine();
```

- InputStreamReader : conversion d'un flux d'octets en flux de caractère
- BufferedReader : gestion de lignes

```
PrintWriter pred = new PrintWriter(
    new BufferedWriter(
          new OutputStreamWriter(
                  socket.getOutputStream())),
    true);
```

- PrintWriter : impression formatée

Socket UDP en lecture

```
try {
  int p = <numéro de port>;
  byte[] t = new byte[<taille>];
  DatagramSocket s = new DatagramSocket(p);
  DatagramPacket d = new DatagramPacket(t,t.length);
  s.receive(d);
  String donnee = new String(d.getData(), 0, t.length);
  System.out.println(d.getAddress()+d.getPort()+donnee);
  …
}
catch (Exception e) {
  System.err.println(e);
}
```

émetteur

Socket UDP en écriture

```
try {
   int p = <numéro de port>;
   byte[] t = new byte[<taille>];
   FileInputStream f = new FileInputStream(…);
   f.read(t);
   DatagramSocket s = new DatagramSocket(p);
   DatagramPacket d = new DatagramPacket(t, t.length,
     InetAddress.getByName("proton.inrialpes.fr"), 8081);
   s.send(d);
   …
} catch (Exception e) {
   System.err.println(e);
}
```

destinataire

Un exemple complet : TCP + serialisation + threads

Passage d'objets par valeur à l'aide de la serialization

L'objet à passer au serveur :

```
public class Voiture implements Serializable {
    public String type;
    public int imm;
    public voiture(String type, int imm) {
        this.type = type;
        this.imm = imm;
    }
    public String toString() {
        return this.type+" "+this.imm;
    }
}
```


Un exemple complet : TCP + serialisation + threads

Le client

```
public class Client {
   public static void main (String[] str) {
      try {
         Socket csock = new Socket("efate",6666);
         ObjectOutputStream oos = new ObjectOutputStream (
                                        csock.getOutputStream());
         oos.writeObject(new voiture("peugeot",171838));
         csock.close();
       } catch (Exception e) {
            System.out.println("An error has occurred ...");
       }
   }
}
```


Un exemple complet : TCP + serialisation + threads

Le serveur

```
public class Server {
   public static void main (String[] str) {
      ServerSocket ss;
      int port = 6666;
      ss = new ServerSocket(port);
      System.out.println("Server ready ...");

      while (true) {
         Slave sl = new Slave(ss.accept());
         sl.start();
      }
   }
}
```


Un exemple complet : TCP + serialisation + threads

L'esclave :

```
public class Slave extends Thread {
   Socket ssock;
   public slave(Socket s) {
      this.ssock = s;
   }
   public void run() {
      ObjectInputStream ois = new ObjectInputStream(
                                 ssock.getInputStream());
      Voiture v = (Voiture)ois.readObject();
      System.out.println("voiture recue : "+ v.toString());
      ssock.close();
   }
}
```

Conclusion

- Programmation à l'aide des sockets
  - Relativement simple
  - Permet un contrôle fin sur les messages échangés
  - Fonction de base
- Services de plus au niveau réutilisables
  - Appel de procédure à distance
  - Messages persistants
  - Diffusion
  - ....

Des tutoriaux de programmation socket en Java plein le Web …

Exemple : https://openclassrooms.com/courses/introduction-aux-sockets-1