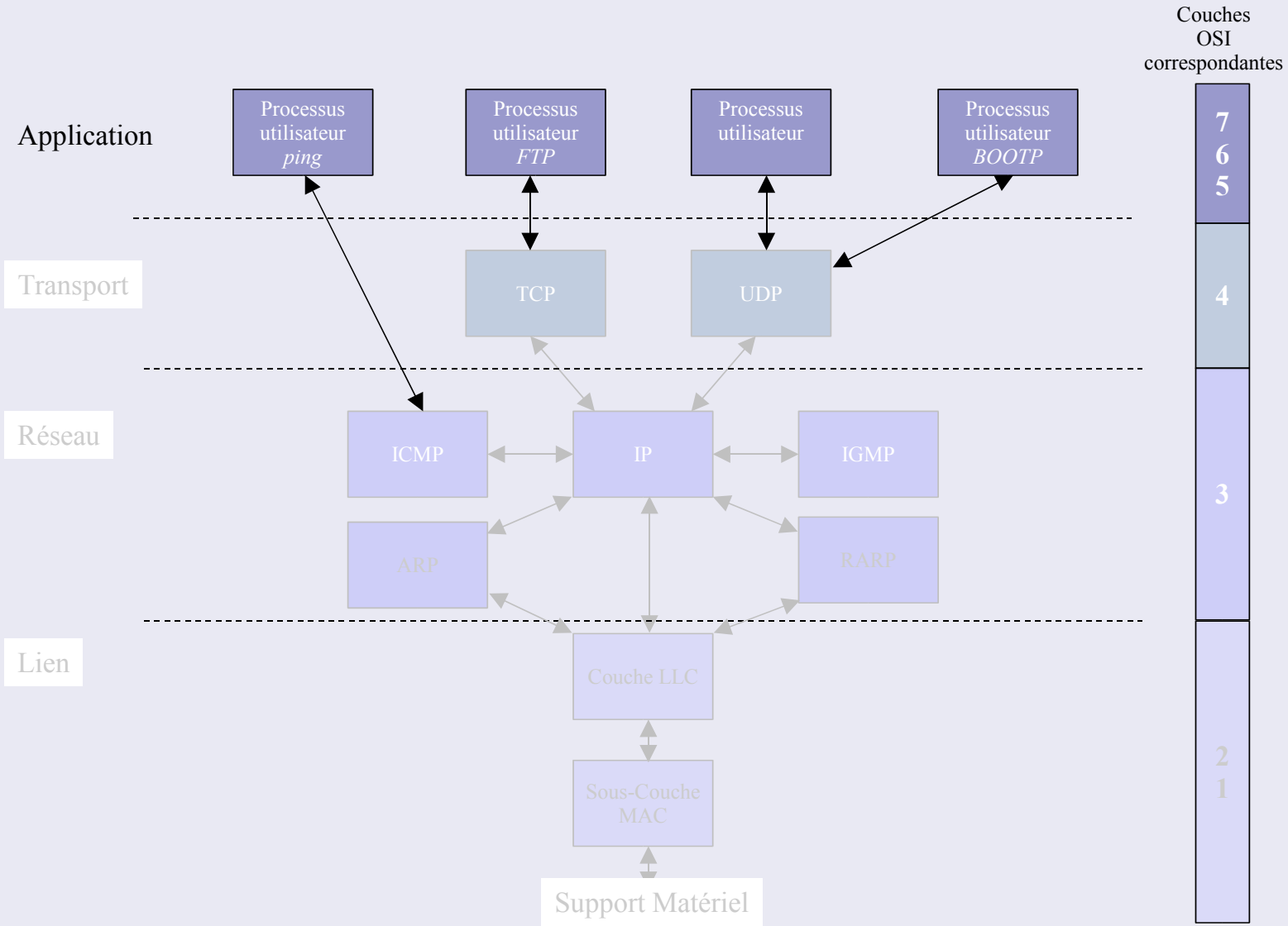
# Programmation socket

Géraldine Del Mondo

Basé sur le cours de M. Nicolas Delestre - Dpt ASI

Architecture des Systèmes d'Information

# Repositionnement



# Le modèle client-serveur 1 / 2

- TCP/IP permet la communication point à point entre deux applications
- TCP/IP ne fournit aucun mécanisme permettant l'exécution automatique d'un programme à l'arrivée d'un message
- Donc dans une communication point à point, l'une des applications doit attendre l'initiative de la communication de la part de l'autre application

# Le modèle client-serveur 2 / 2

**Définition : Client**

Application qui prend l'initiative du lancement de la communication

**Définition : Serveur**

Application qui attend la communication

# Le concept de socket 1 / 2

- Un socket est une “entité” qui permet à deux processus de communiquer
- Au contraire des tubes nommés :
  - Les deux processus n’ont pas besoin d’être sur la même machine
  - Alors que les tubes nommés fournissent deux descripteurs de fichier (un pour la lecture et un pour l’écriture), les sockets proposent une communication bi-directionnelle

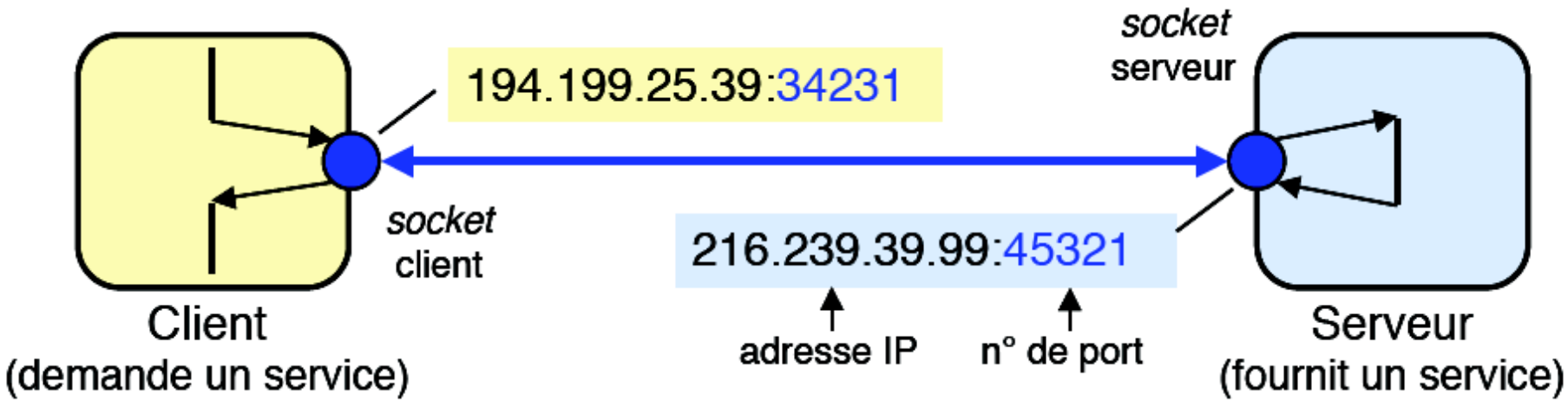



Illustration [1]

# Le concept de socket 2 / 2

- La création d'un socket entre deux processus :
  - crée un "tuyau" persistant virtuel entre les deux processus si la communication est orientée **connexion**
  - crée un mécanisme d'envoi de message simplifié si la communication entre les deux processus est orientée **sans connexion**

# Les sockets sur internet

- Lorsque l'on veut créer un socket sur Internet :
  - on crée un socket s'appuyant sur TCP lorsque l'on veut un mécanisme de communication orientée connexion
  - on crée un socket s'appuyant sur UDP lorsque l'on veut un mécanisme de communication orientée sans connexion
- Remarque :
  - En général on nomme aussi socket le couple adresse IP + port

# Création d'un socket 1 / 2

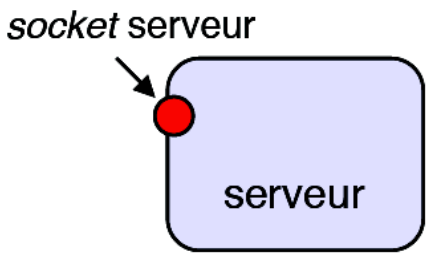
## Les libraries

```
#include <sys/types.h>
#include <sys/socket.h>
```

## Création d'un socket...

```
int socket(int domain, // quelle famille de socket
           int type,   // type de communication
           int protocol)// le protocol
```

→ retourne l'identifiant du socket




Idem côté client

Illustration [1]

## Exemple

```
int s;
s = socket(PF_LOCAL, SOCK_DGRAM, 0);
```

## Tables des correspondances

| Domain | Socket type | Protocol | Description |
|---|---|---|---|
| PF_LOCAL | SOCK_STREAM | 0 | Socket local avec connexion |
| PF_LOCAL | SOCK_DGRAM | 0 | Socket local sans connexion |
| PF_INET | SOCK_STREAM | IPPROTO_TCP | Socket internet basé sur TCP |
| PF_INET | SOCK_DGRAM | IPPROTO_UDP | Socket internet basé sur UDP |

## Attention

- Ne pas confondre PF_xx et AF_xx
- Il y en a d'autres (PF_APPLETALK, SOCK_RAW) : *Voir sys/socket.h*

# Branchement du socket

Une fois que l'on a créé un socket, bind() lui spécifie son adresse :

## bind()

```
int bind(int socketid,          // identifiant socket
         struct sockaddr *addr, // adresse locale
         int addrlen)           // longueur adresse en octets
```

→ retourne -1 si erreur

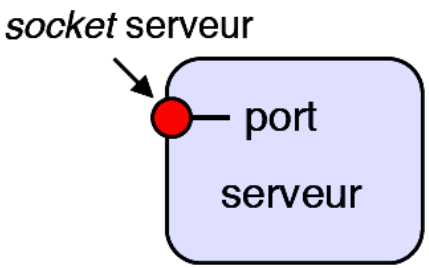



Illustration [1]

Côté client facultatif

# La stucture d'adresse 1 / 3

Sachant que chaque protocole a son format d'adresse, `struct sockaddr` est générique :

## struct sockaddr

```
struct sockaddr {
  u_short sa_family; // AF_LOCAL, AF_INET, etc
  char    sa_data[14]; // l'adresse proprement dite
}
```

Dans la pratique on utilise des variables dont le type correspond au protocole qu'on utilise, par exemple : `struct sockaddr_in`

# La stucture d'adresse 2 / 3

Les deux structures d'adresse les plus utilisées :

## Unix (défini dans *sys/un.h*)

```
struct sockaddr_un {
  sa_family_t sun_family; //AF_LOCAL ou AF_UNIX
  char        sun_path[108];//Le chemin
}
```

## Internet (défini dans *netinet/in.h*)

```
struct sockaddr_in {
  sa_family_t   sin_family; // AF_INET
  in_port_t      sin_port;   // Port TCP ou UDP
  struct in_addr sin_addr; // L'adresse IP
  unsigned char sin_zero[8]; // Inutilisé (souvent à 0)
}
struct in_addr { in_addr_t s_addr }
```

# La stucture d'adresse 3 / 3

## Un exemple

```
struct sockaddr_in
construireAdresseTCPUDPDepuisChaine(char* adresseIP,
                                    char* port) {
  struct sockaddr_in adresse;

  memset(&adresse,0,sizeof adresse);
  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(atoi(port));
  adresse.sin_addr.s_addr = inet_addr(adresseIP); /* ou INADDR_ANY */

  return adresse;
}
```

# Communication orientée sans connexion

- Avantages / Inconvénients
- L'envoi d'un message
- La réception d'un message
- Fonctionnement de base
  - Un exemple de serveur
  - Un exemple de client

# Avantages et inconvénients

- Avantages / aux communications avec connexion
  - Simple : Pas de connexion à établir entre le client et le serveur
  - Rapide
  - Possibilité de broadcaster un message
- Inconvénients :
  - Pas fiable
  - Pas de séquencement des datagrammes
  - Une taille maximale pour les messages

# L'envoi d'un message

## sendto()

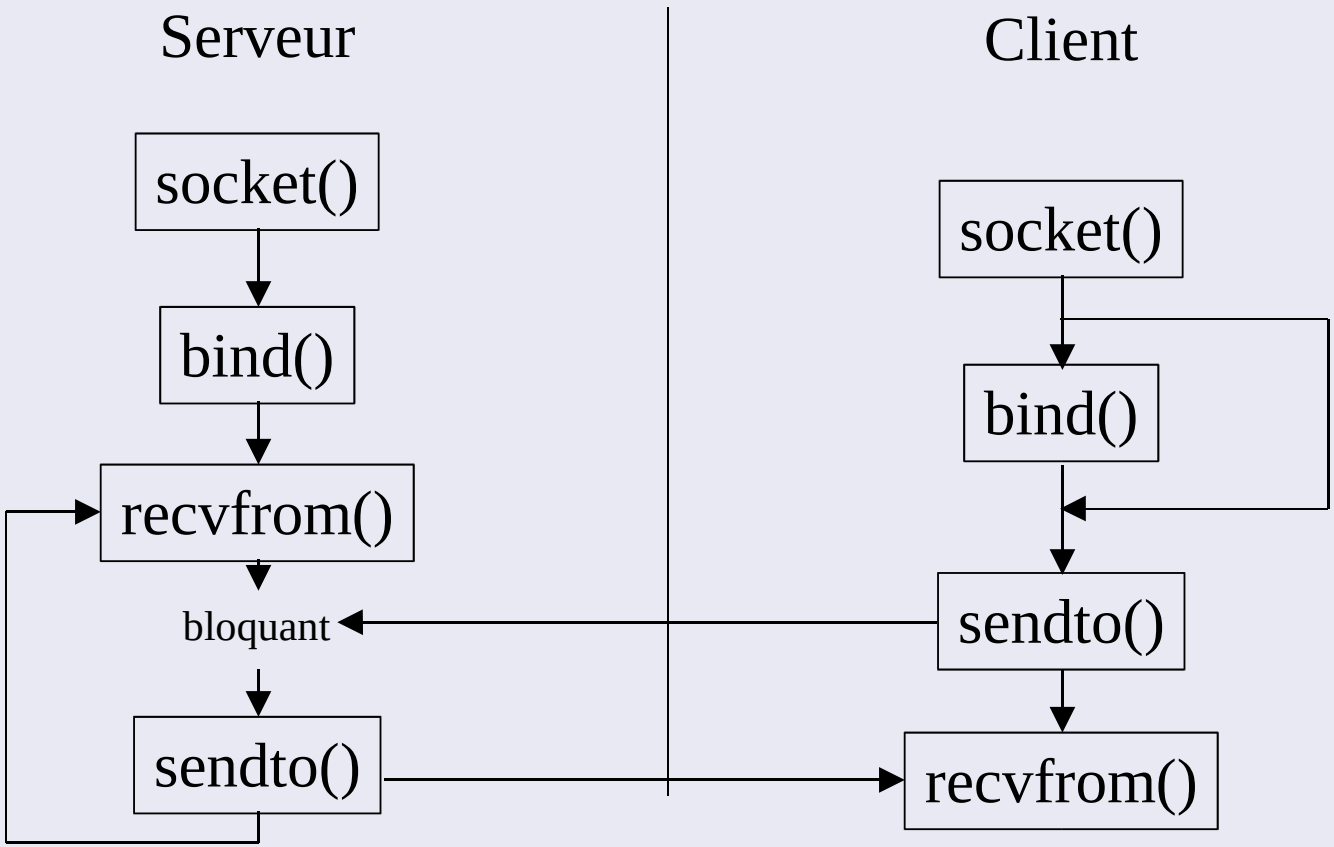
```
int sendto(int s,              // Le socket
           const void *msg, // Le message
           int len,            // Longueur du message
           unsigned flags, // 0 dans 99% des cas
           const struct
             sockaddr *to, // Adresse destinataire
           int tolen);     // longueur de l'adresse
```

# La réception d'un message

## recfrom()

```
int recvfrom(int s,           // Le socket
             void *buf,       // Le buffer pour
                              //  stocker le message
             int len,         // La taille max du
                              //  buffer
             unsigned flags, // 0 dans 99% des cas
             const struct
               sockaddr *from // Adresse de l'envoyeur
             int fromlen);    // Longueur maximale de
                              //  l'adresse
```

# Fonctionnement de base



# Un exemple

- Un programme client qui demande la date et/ou l'heure à un serveur
- Protocole de communication :
  - Le client envoie une chaîne de caractères spécifiant les informations et le format désiré (notation compatible avec la fonction `strftime`) : `%A %b %d %H:%M:%S %Y`
  - Le serveur retourne la chaîne de caractères demandée

## Structure du client : `client.c`

`main()` ⇒ `recupererDateEtHeure()` ⇒ `envoyerRequete()` et `lireResultat()`

## Structure du serveur (version UDP) : `serveur.c`

`main()` ⇒ `ecouter()` ⇒ `traiterRequete()`

## Structure du serveur (version TCP) : `serveur.c`

`main()` ⇒ `ecouter()` ⇒ `traiterRequete()` ⇒ `lireRequete()` et `envoyerResultat()`

# Librairie réseau 1 / 4

## Reseau.h

```
#ifndef __RESEAU_h__
#define __RESEAU_h__
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
struct sockaddr_in
construireAdresseTCPUDPDepuisChaine(char* adresseIP,
                                    char* port);
int creerSocketTCPClient(struct sockaddr_in adresse);
int creerSocketTCPServeur(struct sockaddr_in adresse);
int creerSocketUDPServeur(struct sockaddr_in adresse);
#endif
```

# Librairie réseau 2 / 4

## Reseau.c

```
#include "Reseau.h"

struct sockaddr_in
construireAdresseTCPUDPDepuisChaine(char* adresseIP,
                                    char* port) {
  struct sockaddr_in adresse;

  memset(&adresse,0,sizeof adresse);
  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(atoi(port));
  adresse.sin_addr.s_addr = inet_addr(adresseIP); /* ou INADDR_ANY */

  return adresse;
}
```

# Librairie réseau 3 / 4

## Reseau.c

```
int creerSocketTCPUDPServeur(struct sockaddr_in adresse, int mode) {
  int resultat;
  int longueurAdresse;

  resultat = socket(PF_INET,mode,0);
  if ((resultat == -1) || (adresse.sin_addr.s_addr== INADDR_NONE))
    return -1;
  else {
    longueurAdresse = sizeof adresse;
    if (bind(resultat,
             (struct sockaddr *)&adresse,
             longueurAdresse)!=-1)
      return resultat;
    else
      return -1;
  }
}
```

# Librairie réseau 4 / 4

## Reseau.c

```
int creerSocketUDPServeur(struct sockaddr_in adresse) {
  return creerSocketTCPUDPServeur(adresse,SOCK_DGRAM);
}
```

# Un exemple de serveur UDP 1 / 5

## serveur.c - les inclusions

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "../Reseau.h"

#define TAILLE_BUFFER 128
```

# Un exemple de serveur UDP 2 / 5

## serveur.c - main()

```
int main(int argc,char **argv) {
  char *adresseServeur = NULL;
  char *portServeur = "9099";
  int socketServeur;
  if ( argc >= 2 ) {
    adresseServeur = argv[1];
  } else {
    adresseServeur = "127.0.0.1";
  }
  if ( argc >= 3 )
    portServeur = argv[2];
  socketServeur=
    creerSocketUDPServeur(
      construireAdresseTCPUDPDepuisChaine(adresseServeur,
                                          portServeur));
  if (socketServeur!=-1)
    ecouter(socketServeur);
  return 1;
}
```

# Un exemple de serveur UDP 3 / 5

## serveur.c - ecouter()

```
int ecouter(int socketServeur) {
  int longueurAdresseClient;
  struct sockaddr_in adresseClient;
  int erreur=0;
  char datagram[TAILLE_BUFFER];
  int nbOctets;

  while ((erreur!=-1)) {
    longueurAdresseClient = sizeof adresseClient;
    nbOctets=recvfrom(socketServeur,
                      datagram,
                      sizeof datagram,0 ,
                      (struct sockaddr *)&adresseClient,
                      &longueurAdresseClient);
```

Architecture des Systèmes d'Information

# Un exemple de serveur UDP 4 / 5

## serveur.c - ecouter()

```
    if (nbOctets!=-1) {
      datagram[nbOctets]=0;
      nbOctets=traiterRequete(socketServeur,
                              datagram,
                              adresseClient,
                              longueurAdresseClient);
    } else
      erreur=-1;
  }
  close(socketServeur);
  return erreur;
}
```

# Un exemple de serveur UDP 5 / 5

## serveur.c - traiterRequete()

```
int traiterRequete(int socketServeur,
                   char datagram[],
                   struct sockaddr_in adresseClient,
                   int longueurAdresseClient) {
  time_t heureCourante;
  int nbCaracteres;
  char buffer[TAILLE_BUFFER];

  time(&heureCourante);
  nbCaracteres = (int) strftime(buffer,
                                TAILLE_BUFFER,
                                datagram,
                                localtime(&heureCourante));
  return sendto(socketServeur,
                buffer,strlen(buffer), 0,
                (struct sockaddr *)&adresseClient,
                longueurAdresseClient);
}
```

# Un exemple de client UDP 1 / 4

## client.c - main()

```
int main(int argc,char **argv) {
  char* adresseServeur = NULL;
  char* portServeur = "9099";
  char buffer[TAILLE_BUFFER];
  if ( argc >= 2 ) {
    adresseServeur = argv[1];
  } else {
    adresseServeur = "127.0.0.1";
  }
  if ( argc >= 3 )
    portServeur = argv[2];

  recupererDateEtHeure(adresseServeur,portServeur,buffer);
  printf("%s",buffer);
  return 1;
}
```

# Un exemple de client UDP 2 / 4

## client.c - recupererDateEtHeurer()

```
void recupererDateEtHeure(char* adresseServeur,
                          char* portServeur,
                          char* resultat) {
  struct sockaddr_in addrServeur;
  int socketUDP;
  int nbCaracteres;

  addrServeur=construireAdresseTCPUDPDepuisChaine(adresseServeur,
                                                  portServeur);
  socketUDP= socket(PF_INET,SOCK_DGRAM,0);
  if (socketUDP!=-1) {
    envoyerRequete(socketUDP, addrServeur);
    nbCaracteres=lireResultat(socketUDP, resultat);
    if (nbCaracteres!=-1)
       resultat[nbCaracteres]='\0';
    close(socketUDP);
  } else
    initialiserChaine(resultat);
}
```

# Un exemple de client UDP 3 / 4

## client.c - envoyerRequete()

```
int envoyerRequete(int socket,struct sockaddr_in adresseServeur) {
   char datagram[]="%A %b %d %H:%M:%S %Y\n";
   int longueurAdresseServeur=sizeof adresseServeur;
   return sendto(socket,
          datagram,strlen(datagram),0 ,
          (struct sockaddr *)&adresseServeur,
          longueurAdresseServeur);
}
```

# Un exemple de client UDP 4 / 4

## client.c - lireResultat()

```
int lireResultat(int socket, char* datagram) {
  struct sockaddr_in adresseReponse;
  int longueurAdresseReponse=sizeof adresseReponse;
  return recvfrom(socket,datagram,
                  TAILLE_BUFFER, 0,
                  (struct sockaddr *)&adresseReponse,
                  &longueurAdresseReponse);
}
```

# Communication orientée avec connexion

- Avantages
- La connexion
- Un exemple de client
- Fonctionnement de base d'un serveur :
  - Les fonctions `listen()` et `accept()`
- Un exemple de serveur
- Fonctionnement parallélisé du serveur
- Un autre exemple de serveur

# Avantages

- Une fois que la connexion a été établie, on ne s'occupe pas :
  - des paquets perdus
  - des *timeouts* et des retransmissions
  - paquets dupliqués
- L'analogie avec les fichiers est grande :
  - On établie une connexion avec le socket serveur
  - On peut transmettre des données (pas de limites de taille)
  - On ferme la connexion

# La connexion (client)

## connect()

```
int connect(int socketid,          // Le socket
            struct sockaddr *addr, // Adresse serveur
            int addrlen)           // longueur adresse
```

⇒ Retourne -1 si erreur

⇒ Le *bind* est optionnel

# listen() et accept() (serveur)

## listen()

```
int listen(int socketid, // Le socket
           int backlog) // Longueur file d'attente
```

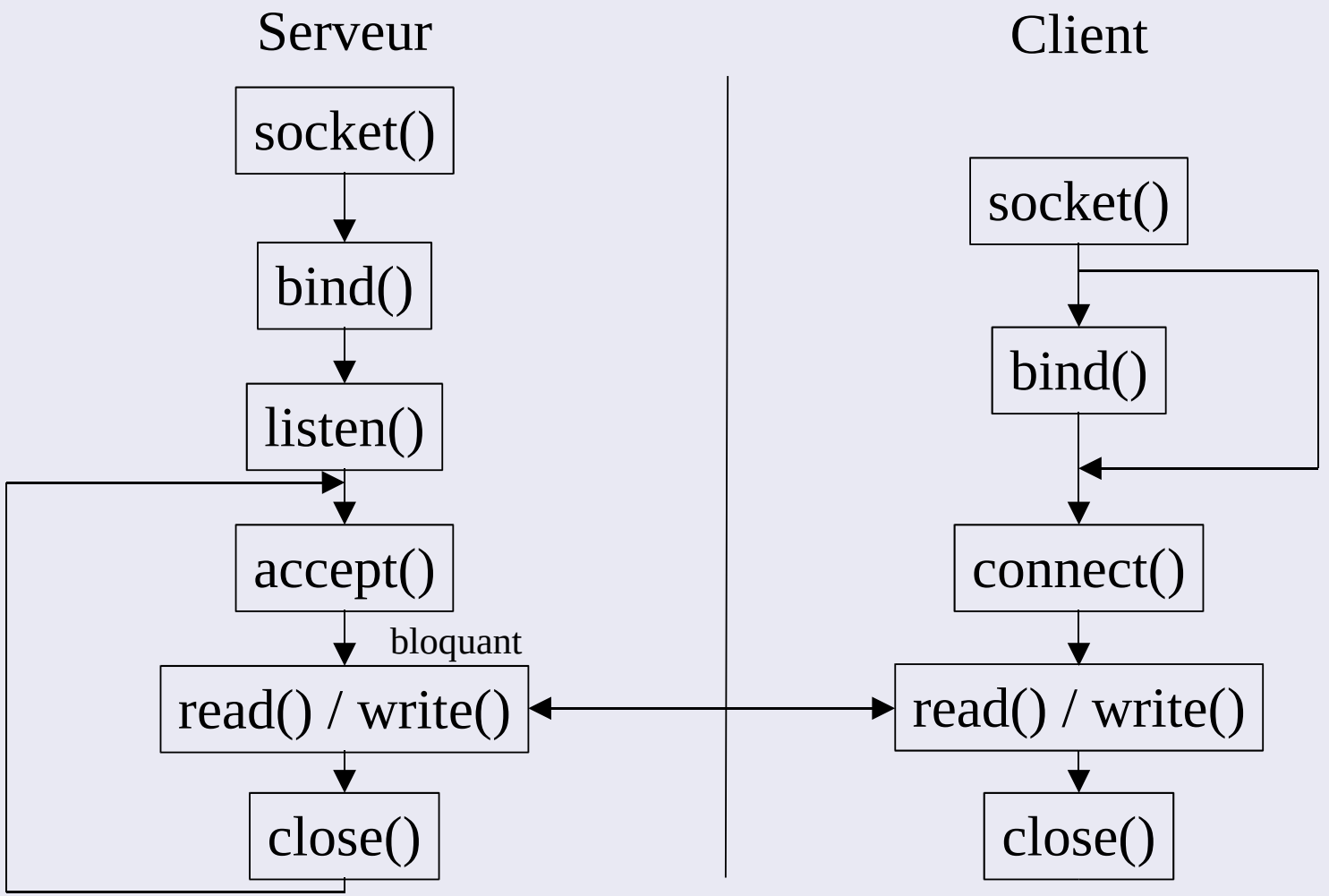
⇒ Retourne -1 si erreur

## accept()

```
int accept(int socketid,        // Le socket
           struct sockaddr* addr,// Adresse du client
           int* addrlen)        // Longueur adresse
```

⇒ Retourne -1 si erreur

# Fonctionnement de base



# Librairie Reseau - Reseau.c (complément) 1 / 2

```
int creerSocketTCPClient(struct sockaddr_in adresse) {
  int resultat;
  int longueurAdresse;

  resultat = socket(PF_INET,SOCK_STREAM,0);
  if ((resultat == -1) || (adresse.sin_addr.s_addr== INADDR_NONE))
    return -1;
  else {
    longueurAdresse = sizeof adresse;
    if (connect(resultat,
                (struct sockaddr *)&adresse,
                longueurAdresse)!=-1)
      return resultat;
    else
      return -1;
  }
}
```

# Librairie Reseau - Reseau.c (complément) 2 / 2

```
int creerSocketTCPServeur(struct sockaddr_in adresse) {
  return creerSocketTCPUDPServeur(adresse,SOCK_STREAM);
}
```

# Un exemple de client TCP 1 / 4

## client.c - main()

```
int main(int argc,char **argv) {
  char* adresseServeur = NULL;
  char* portServeur = "9099";
  char buffer[TAILLE_BUFFER];

  if ( argc >= 2 ) {
    adresseServeur = argv[1];
  } else {
    adresseServeur = "127.0.0.1";
  }
  if ( argc >= 3 )
    portServeur = argv[2];

  recupererDateEtHeure(adresseServeur,portServeur,buffer);
  printf("%s",buffer);
  return 1;
}
```

# Un exemple de client TCP 2 / 4

## client.c - recupererDateEtHeure()

```
void recupererDateEtHeure(char* adresseServeur,
                          char* portServeur,
                          char* resultat) {
  struct sockaddr_in addrServeur;
  int socketTCP;


  addrServeur=construireAdresseTCPUDPDepuisChaine(adresseServeur,
                                                  portServeur);
  socketTCP= creerSocketTCPClient(addrServeur);
  if (socketTCP!=-1) {
    envoyerRequete(socketTCP);
    lireResultat(socketTCP, resultat);
    close(socketTCP);
  } else
    initialiserChaine(resultat);
}
```

# Un exemple de client TCP 3 / 4

## client.c - envoyerRequete()

```
int envoyerRequete(int socket) {
   char formatDateHeure[]="%A %b %d %H:%M:%S %Y\n";
   return write(socket,formatDateHeure,strlen(formatDateHeure));
}
```

# Un exemple de client TCP 4 / 4

## client.c - lireResultat()

```
int lireResultat(int socket, char* buffer) {
  int nbCaracteres=0;
  nbCaracteres=read(socket,buffer,TAILLE_BUFFER-1);
  if (nbCaracteres!=-1)
    buffer[nbCaracteres]='\0';
  return nbCaracteres;
}
```

# Un exemple de serveur TCP 1 / 5

## serveur.c - main()

```
int main(int argc,char **argv) {
  char *adresseServeur=NULL;
  char *portServeur="9099";
  int socketServeur;
  if ( argc >= 2 ) {
    adresseServeur=argv[1];
  } else {
    adresseServeur="127.0.0.1";
  }
  if ( argc >= 3 )
    portServeur=argv[2];

  socketServeur=creerSocketTCPServeur(
     construireAdresseTCPUDPDepuisChaine(adresseServeur,
                                         portServeur));
  if (socketServeur!=-1)
    ecouter(socketServeur,10);
  return 1;
}
```

# Un exemple de serveur TCP 2 / 5

## serveur.c - ecouter()

```
int ecouter(int socketServeur, int longueurFileDAttente) {
  socklen_t longueurAdresseClient;
  struct sockaddr_in adresseClient;
  int socketClient;
  int erreur=0;
```

# Un exemple de serveur TCP 3 / 5

## serveur.c - ecouter()

```
  erreur=listen(socketServeur,longueurFileDAttente);
  while (erreur!=-1) {
      longueurAdresseClient=sizeof adresseClient;
      socketClient=accept(socketServeur,
                          (struct sockaddr *)&adresseClient,
                          &longueurAdresseClient);
      if (socketClient!=-1) {
        if (traiterRequete(socketClient))
          close(socketClient);
        else
          erreur=-1;
      } else
        erreur=-1;
    }
  return erreur;
}
```

# Un exemple de serveur TCP 4 / 5

## serveur.c - traiterRequete()

```
int traiterRequete(int socketClient) {
  char formatDateHeure[TAILLE_BUFFER];
  if (lireRequete(socketClient,formatDateHeure)!=-1)
    return envoyerResultat(socketClient,formatDateHeure);
  else
    return -1;
}
```

# Un exemple de serveur TCP 5 / 5

## serveur.c - lireRequete() et envoyerResultat()

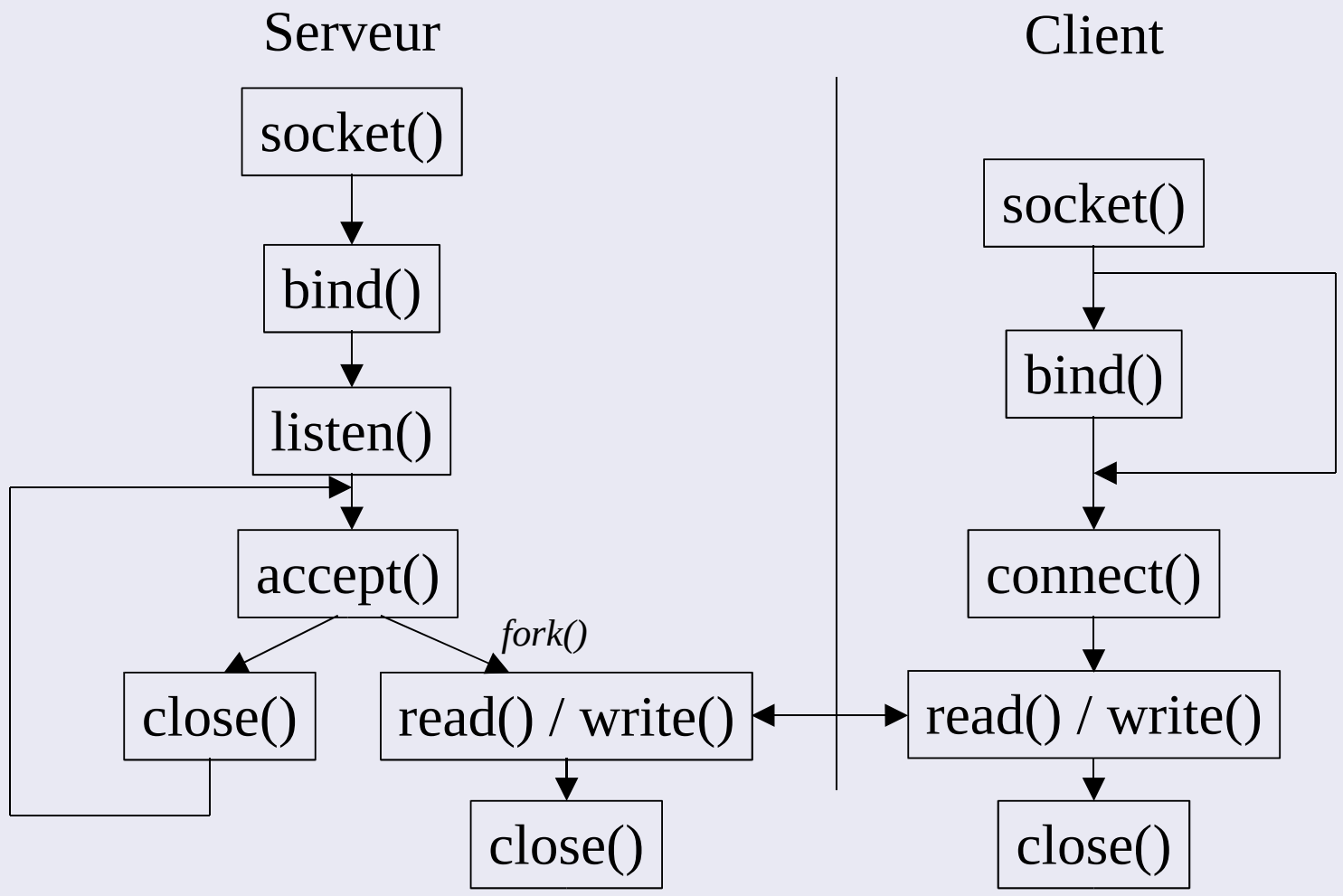
```
int lireRequete(int socket, char formatDateHeure[]) {
  int nbCaracteres=0;
  nbCaracteres=read(socket,formatDateHeure,TAILLE_BUFFER-1);
  if (nbCaracteres!=-1)
    formatDateHeure[nbCaracteres]='\0';
  return nbCaracteres;
}

int envoyerResultat(int socket, char formatDateHeure[]) {
  time_t heureCourante;
  char buffer[TAILLE_BUFFER];
  int nbCaracteres;
  time(&heureCourante);
  nbCaracteres=(int)strftime(buffer,
                             sizeof buffer,formatDateHeure,
                             localtime(&heureCourante));
  return write(socket,buffer,nbCaracteres);
}
```

# Remarque

- Que se passe-t-il lorsque plusieurs clients effectuent des requêtes en même temps ?
  - Certains ne pourront pas se connecter
  - Il faut donc soit :
    - Augmenter la file d'attente : Cela ne résout pas les problèmes des temps d'attente
    - Pouvoir traiter plusieurs requête en même temps : La meilleur solution...

# Fonctionnement parallélisé d'un serveur



# Un exemple de serveur // 1 / 2

## serveur.c - ecouter()

```
int ecouter(int socketServeur,
            int longueurFileDAttente) {
  int longueurAdresseClient;
  struct sockaddr_in adresseClient;
  int socketClient;
  int erreur=0;
  pid_t PID;

  erreur=listen(socketServeur,longueurFileDAttente);
  while (erreur!=-1) {
    longueurAdresseClient=sizeof adresseClient;
    socketClient=accept(socketServeur,
                        (struct sockaddr *)&adresseClient,
                        &longueurAdresseClient);
```

# Un exemple de serveur // 2 / 2

## serveur.c - ecouter()

```
    if (socketClient!=-1) {
      if ((PID=fork())!=-1) {
        if (PID==0) {
          if (traiterRequete(socketClient)) {
            close(socketClient);
            exit(0);
          } else
            erreur=-1;
        } else
          close(socketClient);
      } else
        erreur=-1;
    } else
      erreur=-1;
  }
  return erreur;
}
```

# Utilisation des fonctions de lecture et d'écriture standard

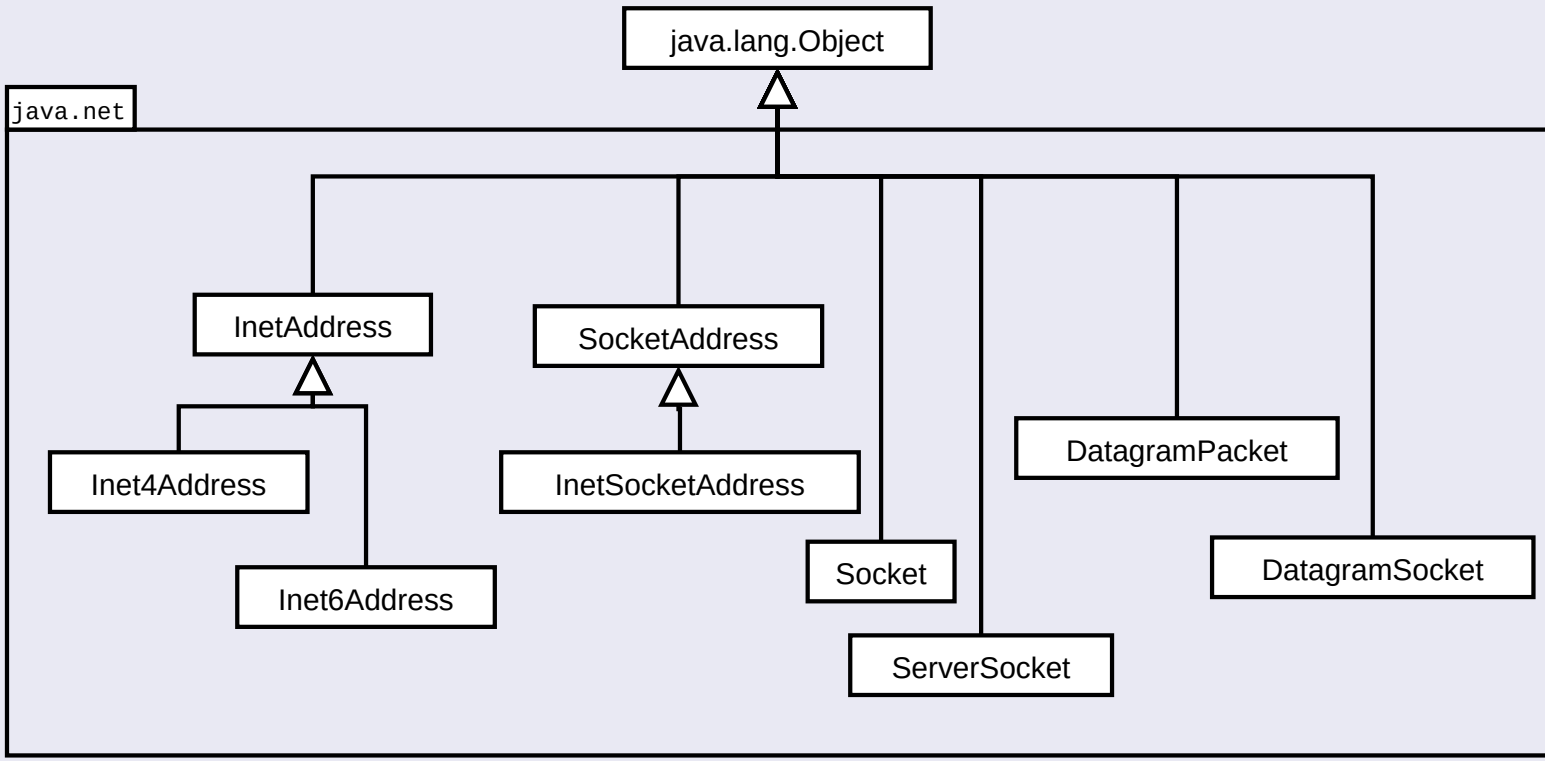
- Sur les fichiers, on a :
  - Les fonctions de base : `open`, `read`, `write`, `close`
  - Les fonctions "étendues" : `fopen`, `fread`, `fwrite`, `fclose`, `fprintf`, `fscanf`, `fgetc`, `fgets`, etc.
- On peut obtenir la même chose sur les sockets de flux :

## fdopen()

```
File* fdopen(int socketid, // Le socket
             char* modes)  // Droits d'accès
```

⇒ Retourne null si erreur

# Le paquetage `java.net`



# La classe `InetAddress`

- Classe permettant de gérer les adresses IP
- On ne peut pas créer d'instance de InetAddress, mais on peut obtenir une instance via les méthodes statiques :
  - `InetAddress getLocalHost()`
  - `InetAddress getByName(String)`
  - `InteAddress[ ] getAllByName(String)`

```
InetAddress localhost=InetAddress.getLocalHost();
```

# La classe Socket

- Classe permettant de créer une socket de flux associé à une adresse IP (String ou InetAdress) et un port

## Méthodes de base

- `InetAddress getLocalAdress()`
- `int getLocalPort()`
- `InetAddress getInetAddress()`
- `int getPort()`
- `void close()`
- `InputStream getInputStream()`
- `OutputStream getOutputStream()`

# La classe `ServerSocket`

- Classe permettant de créer un socket d'écoute en précisant au moins le port d'écoute, ainsi que la longueur de la file d'attente et l'adresse IP associée
- La méthode bloquante `accept()` retourne un socket (instance de la classe `Socket`) dès qu'un client se connecte

# La classe DatagramPacket

- Classe permettant de construire des paquets d'information pour une communication orientée sans connexion
- La création du paquet diffère suivant l'utilisation que l'on veut en faire :
  - `DatagramPacket(byte[] buf, int length)`
  - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`

## Méthodes de base

- `InetAddress getAddress()`
- `byte[] getData()`
- `int getLength()`
- `int getPort()`
- `void setAddress(InetAddress iaddr)`
- `void setData(byte[] buf)`
- `void setLength(int length)`
- `void setPort(int iport)`

Architecture des Systèmes d'Information

# La classe DatagramSocket

- Classe permettant de créer un socket pour des communications orientées sans connexion
- Trois constructeurs disponibles :
  - `DatagramSocket()`
  - `DatagramSocket(int port)`
  - `DatagramSocket(int port,InetAddress addr)`

## Méthodes de base

- `void close()`
- `void connect(InetAddress addr, int port)`
- `void disconnect()`
- `InetAddress getInetAddress()`
- `int getPort()`
- `InetAddress getLocalAddress()`
- `int getLocalPort()`
- `void receive(DatagramPacket p)`
- `void send(DatagramPacket p)`

Architecture des Systèmes d'Information

# Communication TCP - Un exemple de serveur 1 / 5

## Serveur.java - main()

```
public static void main(String[] arguments) {
    ServerSocket socketDEcoute;
    String adresseIPDEcoute="127.0.0.1";
    int portTCPDEcoute=9099;

    switch (arguments.length) {
        case 1 :
            adresseIPDEcoute=arguments[0];
            break;
        case 2 :
            adresseIPDEcoute=arguments[0];
            portTCPDEcoute=new Integer(arguments[1]).intValue();
            break;
    }
```

# Communication TCP - Un exemple de serveur 2 / 5

## Serveur.java - main()

```
        try {
            socketDEcoute=new ServerSocket(portTCPDEcoute,
                    NB_SOCKETS_EN_PARALLELE,
                    InetAddress.getByName(adresseIPDEcoute));
            for (;;) {
                new Thread(new Serveur(socketDEcoute.accept())).start()
                    ;
            }
        } catch (Exception e) {
            System.out.println("Erreur "+e);
        }
    }
```

# Communication TCP - Un exemple de serveur 3 / 5

## Serveur.java - *entête* et *constructeur*

```
package Flux;

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.catalina.util.Strftime;

class Serveur implements Runnable {
    Socket socketClient;
    private static final int NB_SOCKETS_EN_PARALLELE=10;

    Serveur(Socket socketClient) {
        this.socketClient=socketClient;
    }
```

# Communication TCP - Un exemple de serveur 4 / 5

## Serveur.java - run()

```
public void run() {
    try {
        envoyerResultat(lireRequete());
        socketClient.close();
    } catch (Exception e) {
        System.out.println("Erreur avec un client : "+e);
    }
}
```

# Communication TCP - Un exemple de serveur 5 / 5

## Serveur.java - lireRequete() et envoyerResultat()

```
private String lireRequete() throws java.io.IOException {
    BufferedReader fluxEntree=new BufferedReader(
            new InputStreamReader(
            socketClient.getInputStream()));
    String format=fluxEntree.readLine();
    return format;
}

private void envoyerResultat(String format) throws java.io.
     IOException {
    BufferedWriter fluxSortie=new BufferedWriter(
            new OutputStreamWriter(
            socketClient.getOutputStream()));
    fluxSortie.write(new Strftime(format).format(new Date()));
    fluxSortie.flush();
}
```

# Communication TCP - Un exemple de client 1 / 4

## Client.java - main()

```
public static void main(String[] arguments) {
    Client leClient;
    switch (arguments.length) {
        case 0 :
            leClient=new Client();
            break;
        case 1 :
            leClient=new Client(arguments[0]);
            break;
        default :
            leClient=new Client(arguments[0],arguments[1]);
            break;
    }
    leClient.afficherHeure();
}
```

# Communication TCP - Un exemple de client 2 / 4

## Client.java- *entête* et *constructeurs*

```
package Flux;

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

class Client {
    String adresseIPServeur;
    String portTCPServeur;

    Client() {
        this("127.0.0.1","9099");
    }
    Client(String adresseIP) {
        this(adresseIP,"2001");
    }
    Client(String adresseIP, String portTCP) {
        adresseIPServeur=adresseIP;
        portTCPServeur=portTCP;
    }
```

# Communication TCP - Un exemple de client 3 / 4

## Client.java - afficherHeure()

```
void afficherHeure() {
    Socket socket;
    try {
        socket=new Socket(
                adresseIPServeur,new Integer(portTCPServeur).
                    intValue());
        envoyerRequete(socket);
        System.out.println(lireResultat(socket));
        socket.close();
    } catch (Exception e) {
        System.out.println("Erreur : "+e);
    }
}
```

# Communication TCP - Un exemple de client 4 / 4

## Client.java - envoyerRequete() et lireResultat()

```
private void envoyerRequete(Socket socket) throws java.io.
    IOException {
    String formatHeure="%A %b %d %H:%M:%S %Y\n";
    BufferedWriter fluxSortie;
    fluxSortie=new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream()));
    fluxSortie.write(formatHeure);
    fluxSortie.newLine();
    fluxSortie.flush();
}

private String lireResultat(Socket socket) throws java.io.
    IOException {
    BufferedReader fluxEntree;
    fluxEntree=new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
    return fluxEntree.readLine();
}
```

# Communication UDP - Un exemple de serveur 1 / 5

## Serveur.java - main()

```
public static void main(String[] arguments) {
    String adresseIPDEcoute="127.0.0.1";
    int portUDPDEcoute=9099;
    switch (arguments.length) {
        case 1 :
            adresseIPDEcoute=arguments[0];
            break;
        case 2 :
            adresseIPDEcoute=arguments[0];
            portUDPDEcoute=new Integer(arguments[1]).intValue();
            break;
    }
    try {
        new Serveur(InetAddress.getByName(adresseIPDEcoute),
                portUDPDEcoute).traiterRequetes();
     } catch (Exception e) {
        System.out.println("Erreur d'adresse : "+e);
    }
}
```

# Communication UDP - Un exemple de serveur 2 / 5

## Serveur.java - *entête* et *constructeur*

```
package Datagram;

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.catalina.util.Strftime;

public class Serveur  {
    private InetAddress adresseIPDEcoute;
    private int portUDPDEcoute;

    private Serveur(InetAddress adresseIPDEcoute,
            int portUDPDEcoute) {
        this.adresseIPDEcoute=adresseIPDEcoute;
        this.portUDPDEcoute=portUDPDEcoute;
    }
```

# Communication UDP - Un exemple de serveur 3 / 5

## Serveur.java - traiterRequetes()

```
private void traiterRequetes() {
    DatagramSocket socket;
    DatagramPacket paquetUDPClient;
    try {
        socket=new DatagramSocket(portUDPDEcoute,adresseIPDEcoute);
        while (socket != null) {
            paquetUDPClient=lireRequete(socket);
            envoyerResultat(new String(paquetUDPClient.getData(),
                        0,
                        paquetUDPClient.getLength()),
                    socket,
                    paquetUDPClient.getAddress(),
                    paquetUDPClient.getPort()
                );
        }
        socket.close();
    } catch (Exception e) {
        System.out.println("Erreur avec un client : "+e);
    }
```

# Communication UDP - Un exemple de serveur 4 / 5

## Serveur.java - lireRequete()

```
private DatagramPacket lireRequete(DatagramSocket socket)
    throws java.io.IOException {
    DatagramPacket paquetUDP = new DatagramPacket(new byte[512],
        512);
    socket.receive(paquetUDP);
    return paquetUDP;
}
```

# Communication UDP - Un exemple de serveur 5 / 5

## Serveur.java - envoyerResultat()

```
private void envoyerResultat(String format,
        DatagramSocket socket,
        InetAddress adresseIPClient,
        int portUDPClient)
    throws java.io.IOException {
    String aEnvoyer=new Strftime(format).format(new Date());
    byte[] buffer=aEnvoyer.getBytes();
    DatagramPacket paquetUDP=new DatagramPacket(buffer,
            buffer.length,
            adresseIPClient,
            portUDPClient);
    socket.send(paquetUDP);
}
```

# Un exemple de client 1 / 5

## Client.java - main()

```
public static void main(String[] arguments) {
   Client leClient;
    switch (arguments.length) {
        case 0 :
            leClient=new Client();
            break;
        case 1 :
            leClient=new Client(arguments[0]);
            break;
        default :
            leClient=new Client(arguments[0],arguments[1]);
            break;
    }
    leClient.afficherHeure();
}
```

# Un exemple de client 2 / 5

## Client.java - *entête* et *constructeurs*

```
package Datagram;

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

public class Client {
    String adresseIPServeur;
    String portUDPServeur;

    Client() {
        this("127.0.0.1","9099");
    }
    Client(String adresseIP) {
        this(adresseIP,"2001");
    }
    Client(String adresseIP, String portUDP) {
        adresseIPServeur=adresseIP;
        portUDPServeur=portUDP;
    }
```

# Un exemple de client 3 / 5

## Client.java - afficherHeure()

```
private void afficherHeure() {
    DatagramSocket socket;
    try {
        socket = new DatagramSocket();
        envoyerRequete(socket);
        System.out.println(lireResultat(socket));
        socket.close();
    } catch (Exception e) {
        System.out.println("Erreur : "+e);
    }
}
```

# Un exemple de client 4 / 5

## Client.java - envoyerRequete()

```
private void envoyerRequete(DatagramSocket socket) throws
    java.io.IOException,java.net.UnknownHostException {

    String formatHeure="%A %b %d %H:%M:%S %Y\n";
    byte[] buffer=formatHeure.getBytes();
    DatagramPacket paquetUDP=new DatagramPacket(buffer,
            buffer.length,
            InetAddress.getByName(adresseIPServeur),
            new Integer(portUDPServeur).intValue());

    socket.send(paquetUDP);
}
```

# Un exemple de client 5 / 5

## Client.java - lireResultat()

```
private String lireResultat(DatagramSocket socket) throws
    java.io.IOException {
    DatagramPacket paquetUDP=new DatagramPacket(new byte[512], 512)
        ;
    socket.receive(paquetUDP);
    String resultat=new String(paquetUDP.getData(),0,paquetUDP.
        getLength());
    return resultat;
}
```

# Références

[1] Olivier Dalle,
Communication par sockets, `http://deptinfo.unice.fr/~dalle/wiki/uploads/Enseignements/7-SYS-L2.pdf`