Les Sockets

- Généralités
- Programmation
  - En langage C
  - En Java

Généralités (1)

- Mécanisme d'interface de programmation
  - permet aux processus d'échanger des données
  - n'implique par forcément une communication par le réseau

- Une connexion est entièrement définie sur chaque machine par :
  - le type de protocole (TCP, UDP,…)
  - l'adresse IP
  - le numéro de port associé au processus

    → (statiquement pour le serveur, dynamiquement pour le client)

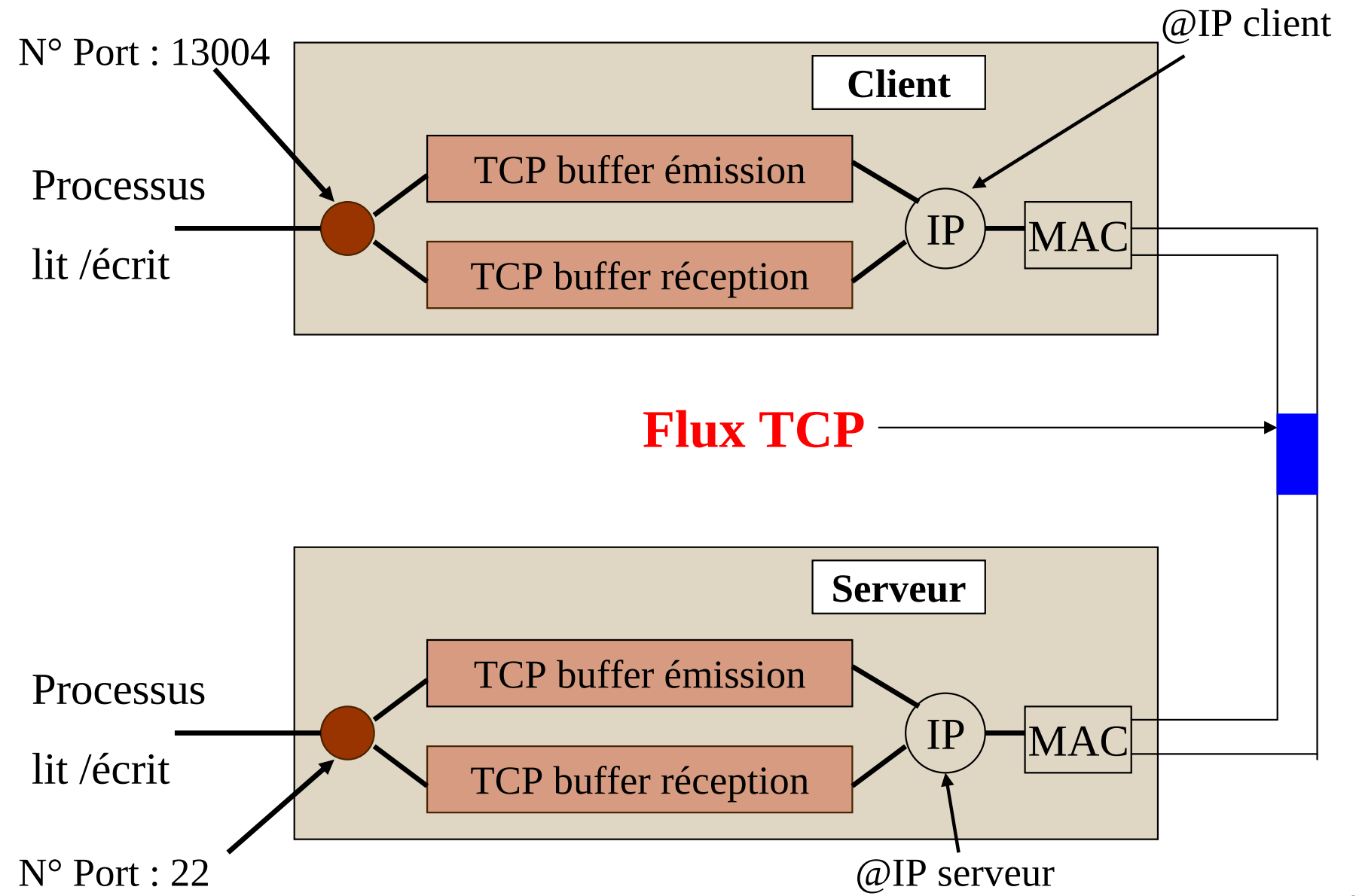
Mode connecté/ non connecté

- Mode connecté (TCP)
  - Problèmes de communication gérés automatiquement
  - Gestion de la connexion coûteuse en message
  - Primitives simples d'émission et de réception
  - Pas de délimitation des messages dans le tampon

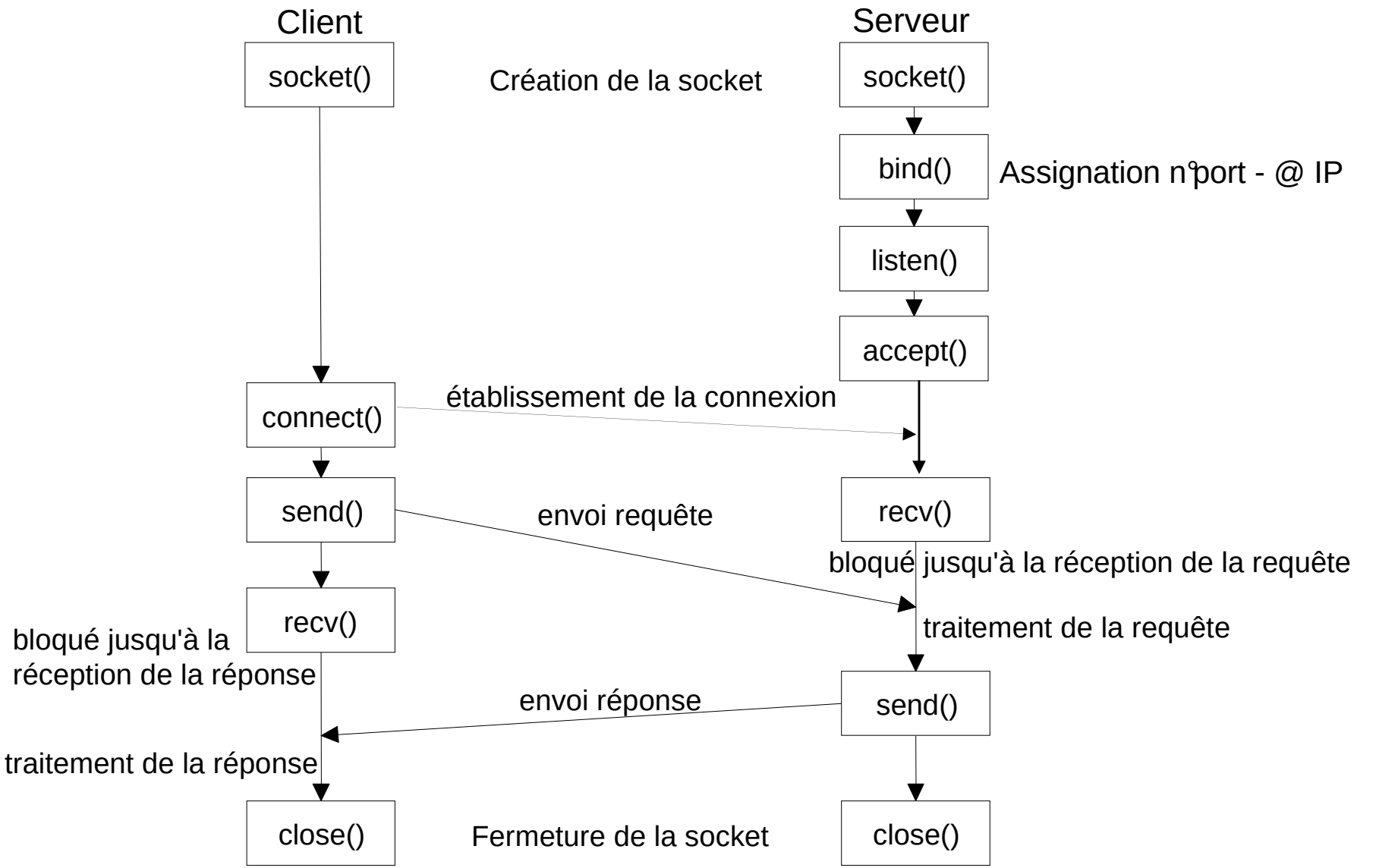
- Mode non connecté (UDP)
  - Consomme moins de ressources
  - Permet la diffusion
  - Aucun gestion des erreurs,

    ➡ c'est la couche applicative qui doit gérer ce problème

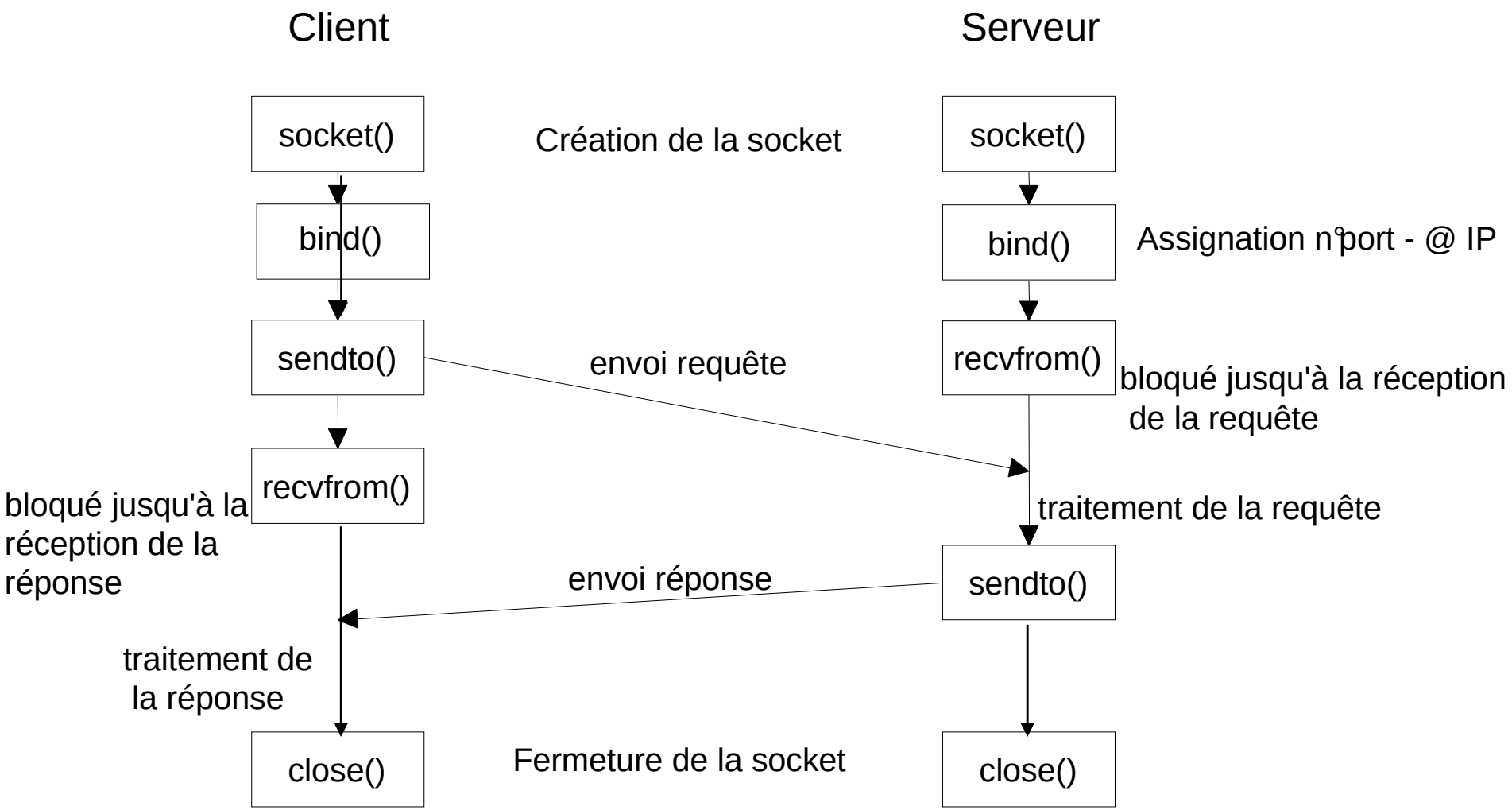
Une connexion TCP



Mode Connecté



Mode non connecté



Généralités (2)

- Une connexion
  - @IP source, @IP destination, port source, port destination
- Une socket est un fichier virtuel sous Unix avec les opérations d'ouverture, fermeture, écriture, lecture …
  (concept légèrement différents sous windows)
- Ces opérations sont des appels systèmes
- Il existe différents types de sockets:
  - Stream socket (connection oriented) ➔ SOCK_STREAM
    mode connecté = TCP
  - Datagram sockets (connectionless) ➔ SOCK_DGRAM
    mode non connecté = UDP
  - Raw sockets ➔ SOCK_RAW
    accès direct au réseau (IP, ICMP)

Programmation en C (1)

- Définition d'une socket
  - **int s = socket (domaine, type, protocole)**
    - *Domaine*
      - AF_UNIX : communication interne à la machine
        structure sockaddr_un dans <sys/un.h>
      - **AF_INET** : communication sur internet en utilisant IP
    - *Type*
      - **SOCK_STREAM** : mode connecté (TCP)
      - **SOCK_DGRAM** : mode non connecté (UDP)
      - SOCK_RAW : utilisation directe niveau 3 (IP, ICMP,…)
    - *Protocole*
      - Actuellement non utilisé, valeur = 0.

Programmation en C (2)

Attachement d'une socket

- **int error = bind (int sock, struct sockaddr *p, int lg)**
  - *error :* entier qui contient le compte-rendu de l'instruction
    - 0 : opération correctement déroulée
    - -1 : une erreur est survenue (en général, problème dans p)
  - *sock :* descripteur de la socket
  - *p :* pointeur vers la zone contenant l'adresse de la station
  - *lg :* longueur de la zone p

Fonction définit avec sockaddr,
→ utilisation réelle de sockaddr_in (AF_INET) ou sockaddr_un (AF_UNIX)

Programmation en C (3)

- Adressage (inclure fichier <sys/socket.h> et <netinet/in.h>)
  - **struct sockaddr_in {**
    short sin_family; ← domaine
    u_short sin_port; ← n° port
    struct in_addr sin_addr; } ← @IP système
  - *sin_port* = numéro de port
    - soit laissé libre
    - soit affecté : htons (n°port) (conversion short -> réseau)
  - *Struct in_addr sin_addr*
    - Si serveur sin_addr.s_addr = INADDR_ANY
    - Si client

```
struct hostent *hp;
hp = gethostbyname(" …. ");
bcopy(hp->h_addr, &x.sin_addr, hp->h_length);
```

Programmation en C (4)

Primitives du serveur

- **int error = listen (int sock, int taille)**
  - *error :* entier qui contient le compte-rendu de l'instruction
    - 0 : opération correctement déroulée , -1 erreur
  - *taille* : nombre de requêtes maximum autorisée

→ Permet de créer une file d'attente pour recevoir les demandes de connexion qui n'ont pas encore été prises en compte

- **int scom = accept (int sock, struct sockaddr *p, int *lg)**
  - *struct sockaddr :* stockage de l'adresse appelant
  - *lg* : longueur de l'adresse appelant
  - *scom* : nouvelle socket permettant la

→ **Primitive bloquante**

Programmation en C (5)

- Ouverture d'une connexion
  - **int error = connect (int sock, struct sockaddr *p, int lg)**
    - *error :* 0 : opération correctement déroulée , -1 erreur
    - *struct sockaddr* : adresse et port de la machine distante
    - *lg* : taille de l'adresse de la machine distante

- Fermeture d'une connexion
  - **int close (int sock)**
    → le plus utilisé
  - **int shutdown (int sock, int how)**
    - *how* : 0 : réception désactivé,
      1 : émission désactivé,
      2 : socket désactivé

Programmation en C (6)

- Emission de données en mode connecté
  - **int send (int sock, char *buf, int lg, int option)**
    - *option :* 0 rien, MSG_OOB pour les messages urgents
  - **int write (int sock, char *buf, int lg)**
    - utilisable seulement en mode connecté, permet d'écrire dans un descripteur de fichier
- Emission de données en mode non connecté
  - **int sendto (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**
    - *p :* contient l'adresse du destinataire
    - *lg_addr :* longueur de l'adresse destinataire

Programmation en C (7)

- Réception de données en mode connecté
  - **int recv (int sock, char *buf, int lg, int option)**
    - *option :* 0 rien, MSG_OOB pour les messages urgents, MSG_PEEK lecture des données sans les retirer de la file d'attente
  - **int read (int sock, char *buf, int lg)**
    - utilisable seulement en mode connecté, renvoie le nombre d'octets réellement lus.
- Reception de données en mode non connecté
  - **int recvfrom (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**
    - *p :* contient l'adresse de la source
    - *lg_addr :* longueur de l'adresse source

Programmation en C (7)

- Quelques fonctions
  - **int getsockname (int sock, struct sockaddr *p, int lg)**
    - permet de récupérer l'adresse locale d'une socket, et son numéro de port
    - ntohs(p->sin_port) pour afficher le numéro du port
    - inet_ntoa(p->sin_addr.s_addr) pour afficher le nom de la machine

- Fonctions bloquantes/ non-bloquantes
  - **fcntl(int sock, F_SETFL, O_NONBLOCK)**
  - **ioctl (int sock, FIONBIO, 0/1)** → (0 : bloquant, 1, non bloquant)
    - permet de rendre une socket bloquante ou non bloquante en lecture/écriture
    - si pas de données en lecture, renvoie err=-1 et errno=EWOULDBLOCK

Programmation en C (8)

- Programmation sous windows
  - **Identique sauf obligation d'initialiser la librairie qui gère les sockets :**

```
#include <winsock.h>
WSADATA info;

if (WSAStartup(MAKEWORD(2,0), &info) == SOCKET_ERROR){
        cout << "erreur dans l'initialisation " <<endl;
        WSACleanup();
        exit(1);
   }
```

Programmation en java (1)

- Création d'une socket en mode connecté et connexion
  - **Côté client**
    - *Socket sock = new socket (nom_serveur, n°port)*
      - permet la connexion direct en TCP
      - plus de recherche de nom
  - **Côté serveur**
    - *ServerSocket sock = new ServerSocket (n°port)*
      Socket scom = sock.accept()
      - la communication s'établit avec la socket scom
      - Plus besoin d'attachement, c'est automatique

Programmation en java (2)

- Lecture/ ecriture sur des sockets en mode connecté

  plusieurs possibilités

  - **cas possible pour une socket soc**
    - *lecture*

      Reader reader = new InputStreamReader(soc.getInputStream())
      BufferedReader texte = new BufferedReader(reader);
      line = texte.readLine();

    - *écriture*

      Printstream sortie = new PrintStream(sock.getOutputStream());
      sortie.println(line);

Programmation en java (3)

- Création d'une socket en mode non connecté
  - **Côté client**
    - *DatagramSocket sock = new DatagramSocket ()*
  - **Côté serveur**
    - *DatagramSocket sock = new DatagramSocket (n°port)*
  - **Emission/réception**
    - *DatagramPacket msg = new DatagramPacket(buffer, taille, serveur, n°port)*
      sock.sent(msg);
    - *DatagramPacket recu = new DatagramPacket(buffer, taille);*
      sock.receive(recu);
      recu.getAddress() -> adresse de l'expéditeur
      recu.getPort() -> N° port de l'expéditeur
      recu.getData() -> les données.

Programmation en java (4)

Utilisation de la classe InetAdress

- **Récupération de l'adresse IP de la machine**
  - *InetAddress a = InetAddress.getLocalHost();*
  - *InetAddress a = InetAddress.getByName("….");*

    - getHostName() : nom de la machine
    - getHostAddress() : numéro IP de la machine
    - toString() : affiche les deux informations précédentes.