Sockets et leur mise en œuvre en C

Les modes de connexion

Le mode *non connecté :* ne garantit pas :

- l'intégrité des données
- l'ordonnancement des données
- la non-duplication des données

❑Ces propriétés doivent être garanties par l'application.

Le mode *connecté :*

- garantit les propriétés ci-dessus
- implique une diminution des performances brutes par rapport au mode non-connecté.
- Permet une implémentation asynchrone des échanges

Les clients

- Une application cliente est moins complexe qu'une application serveur.
- La plupart des applications clientes ne gèrent pas d'interactions avec plusieurs serveurs.
- La plupart des applications clientes sont traitées comme un processus conventionnel ; un serveur peut nécessiter des accès privilégiés.

Les serveurs

Le processus serveur :

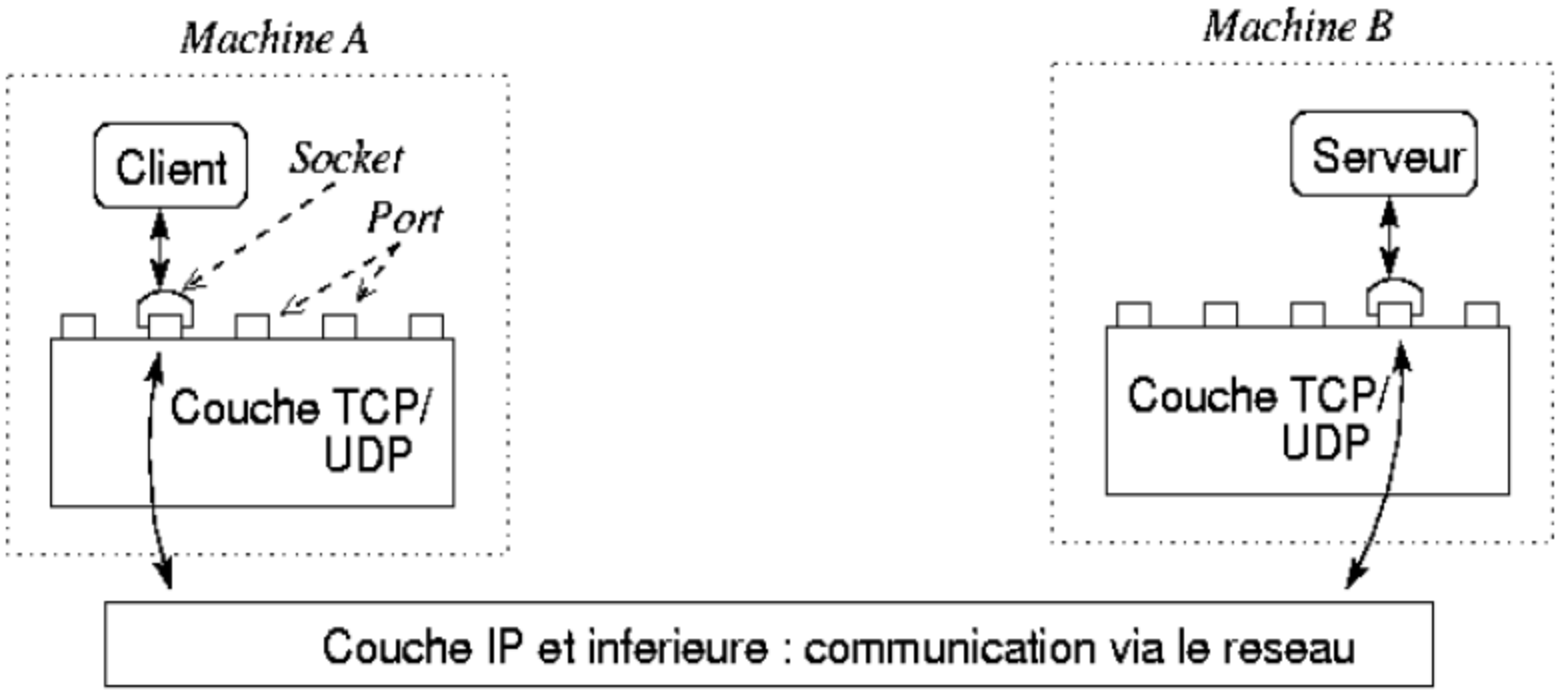
- offre un point d'entrée sur le réseau
- entre dans une boucle infinie d'attente de requêtes
- à la réception d'une requête, déclenche les processus associés à la requête, puis émet la réponse vers le client

- Deux types de serveurs :

✓ itératifs : ne gèrent qu'un seul client à la fois

✓ parallèles : fonctionnent en mode concurrent

Les sockets



Une socket est
✓Un point d'accès aux couches réseau TCP/UDP
✓Liée localement à un port
✓Adressage de l'application sur le réseau : son couple @IP:port
✓Elle permet la communication avec un port distant sur une machine distante : c'est-à-dire avec une application distante

Définition des sockets

Socket : prise

- Associée, liée localement à un port
- C'est un point d'accès aux couches réseaux
- Services d'émission et de réception de données sur la socket via le port
- En mode connecté (TCP)

✓ Connexion = tuyau entre 2 applications distantes

✓ Une socket est un des deux bouts du tuyau

✓ Chaque application a une socket locale pour gérer la communication à distance

- Une socket peut-être liée

✓ Sur un port précis à la demande du programme

✓ Sur un port quelconque libre déterminé par le système

✓ Par défaut, on ne peut lier qu'une socket par port

Principes de bases

- Les sockets permettent l'échange de messages entre 2 processus, situés sur des machines différentes

1- Chaque machine crée une socket,

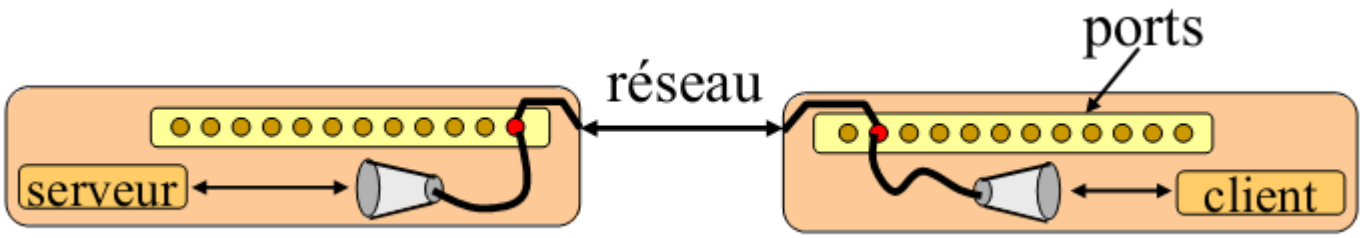
2- Chaque socket sera associée à un port de sa machine hôte,

3- Les deux sockets seront explicit ement connectées si on utilise un protocole en mode connecté …,

4- Chaque machine lit et/ou écrit dans sa socket,

5- Les données vont d'une socket à une autre à travers le réseau,

6- Une fois terminé chaque machine ferme sa socket.



Caractéristiques des sockets

Une socket possède trois caractéristiques :

- **Type:** Décrit la manière dont les données sont transmises : SOCK_DGRAM, SOCK_STREAM, SOCK_RAW.
- **Domaine:** Définit le nommage des sockets et les formats d'adressage utilisés : AF_UNIX, AF_INET, AF_INET6.
- **Protocole:** Décrit le protocole de communication à utiliser pour émettre et recevoir les données. Généralement déterminé par le domaine : PF_UNIX, PF_INET, PF_INET6.

Les domaines des sockets

- Le domaine d'une socket permet de spécifier le mode d'adressage et l'ensemble des protocoles utilisables par la socket.

- Les domaines d'adressage les plus courants sont Unix (AF_UNIX) et IP (AF_INET), voire IPv6 (AF_INET6).

Les domaines des sockets

• Exemples de familles de sockets :
- processus sur la même station Unix :
→ sockets locales (AF_UNIX)
- processus sur des stations diffé rentes à travers Internet :
→ sockets Internet (AF_INET)
• Exemples de modes de communication :
- Datagrames – ou mode non connecté (SOCK_DGRAM)
- Flux de données – ou mode connecté (SOCK_STREAM)
• Exemples de protocoles de sockets :IP, UDP, TCP,

Adressage

- Lorsqu'un processus serveur veut fournir des services, il a besoin d'un port d'attache pour la localisation de ses services.
- Un certain nombre de numéros est réservé aux services standards.
- Pour le serveur, les ports inférieurs à 1024 ne peuvent être utilisés que par le super-utilisateur root.
- Les numéros de port standard sont définis dans le fichier /etc/services, et accessibles via la commande getservbyname().

Structures de socket

- Les structures sockaddr et sockaddr_in sont définies comme suit :

```
struct sockaddr {
        sa_family_t     sa_family;      /* domaine
        char            sa_data[14];    /* adresse
}

struct sockaddr_in {
  sa_family_t           sin_family;     /* domaine
  unsigned short int    sin_port;       /* numro de
  struct in_addr        sin_addr;       /* adresse
  unsigned char         __pad[__SOCK_SIZE__ - ...]
}
```

Création d'une socket

- On crée une socket avec la primitive socket() :

s = socket (AF_INET, SOCK_STREAM, 0);

- s est l'identificateur de la socket. On peut l'utiliser pour les opérations de lecture/écriture.

Gestion des connexions

- bind() permet d'associer une adresse à une socket :

int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)

- connect() est utilisée par le client pour établir la connexion :

int connect(int sockfd, const struct sockaddr,*serv_addr, socklen_t addrlen);

- listen() est utilisée par le serveur pour établir la connexion :

int listen(int s, int backlog);

Gestion des connexions (suite)

- accept() est utilisée par le serveur pour indiquer qu'il est prêt à accepter des connexions. Appel généralement bloquant :

int accept(int s, struct sockaddr *addr, socklen_t *addrlen);

- shutdown() et close() permettent de fermer une connexion :

int shutdown(int s, int how);

Transfert en mode connecté

- Les commandes send() et recv() permettent d'échanger des données :

int send(int s, const void *msg, size_t len, int flags);
int recv(int s, void *buf, size_t len, int flags);

- On peut préciser des options dans les champs flags. Par exemple, MSG_OOB pour les données urgentes.

- Il est également possible d'utiliser les appels read() et write() qui offre moins de possibilités de contrôle.

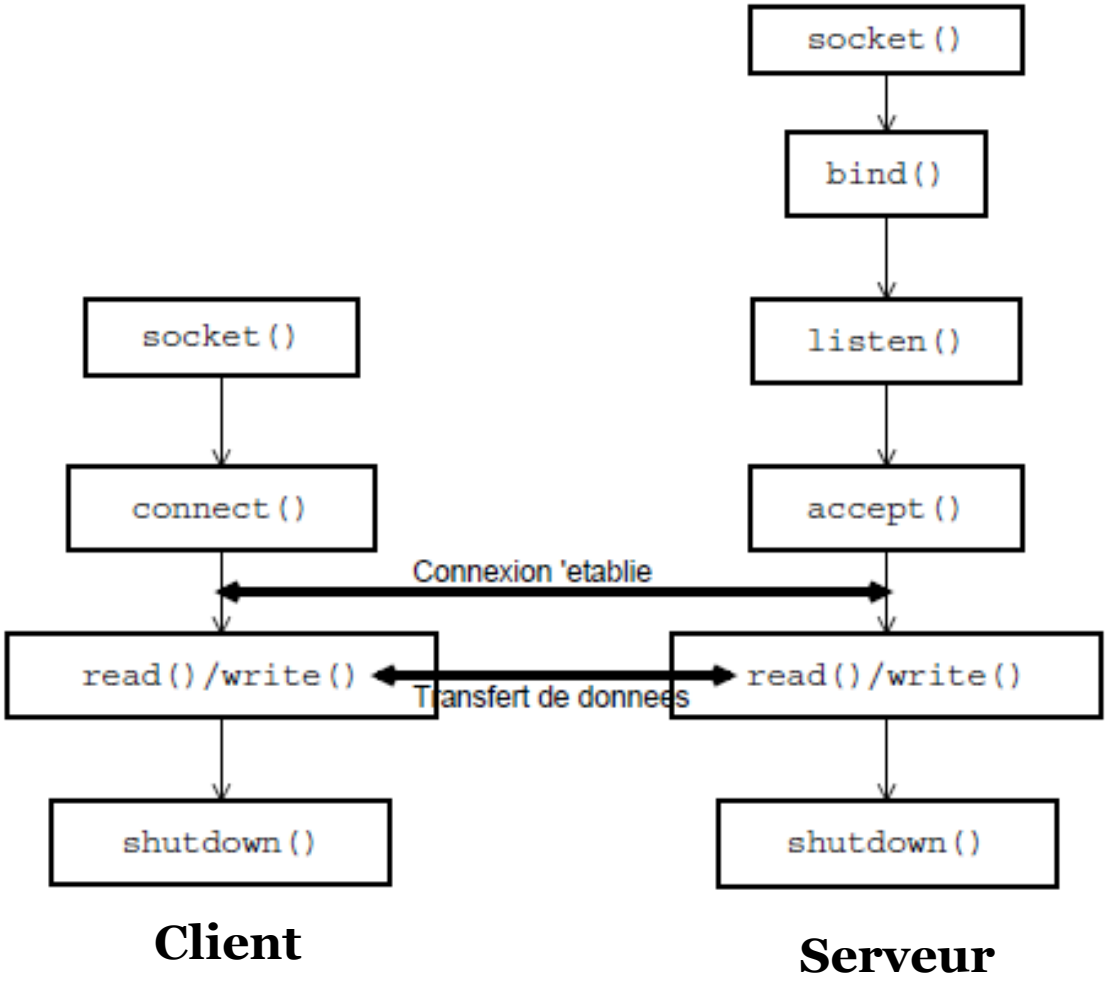
Client-serveur en mode connecté

Principe de communication

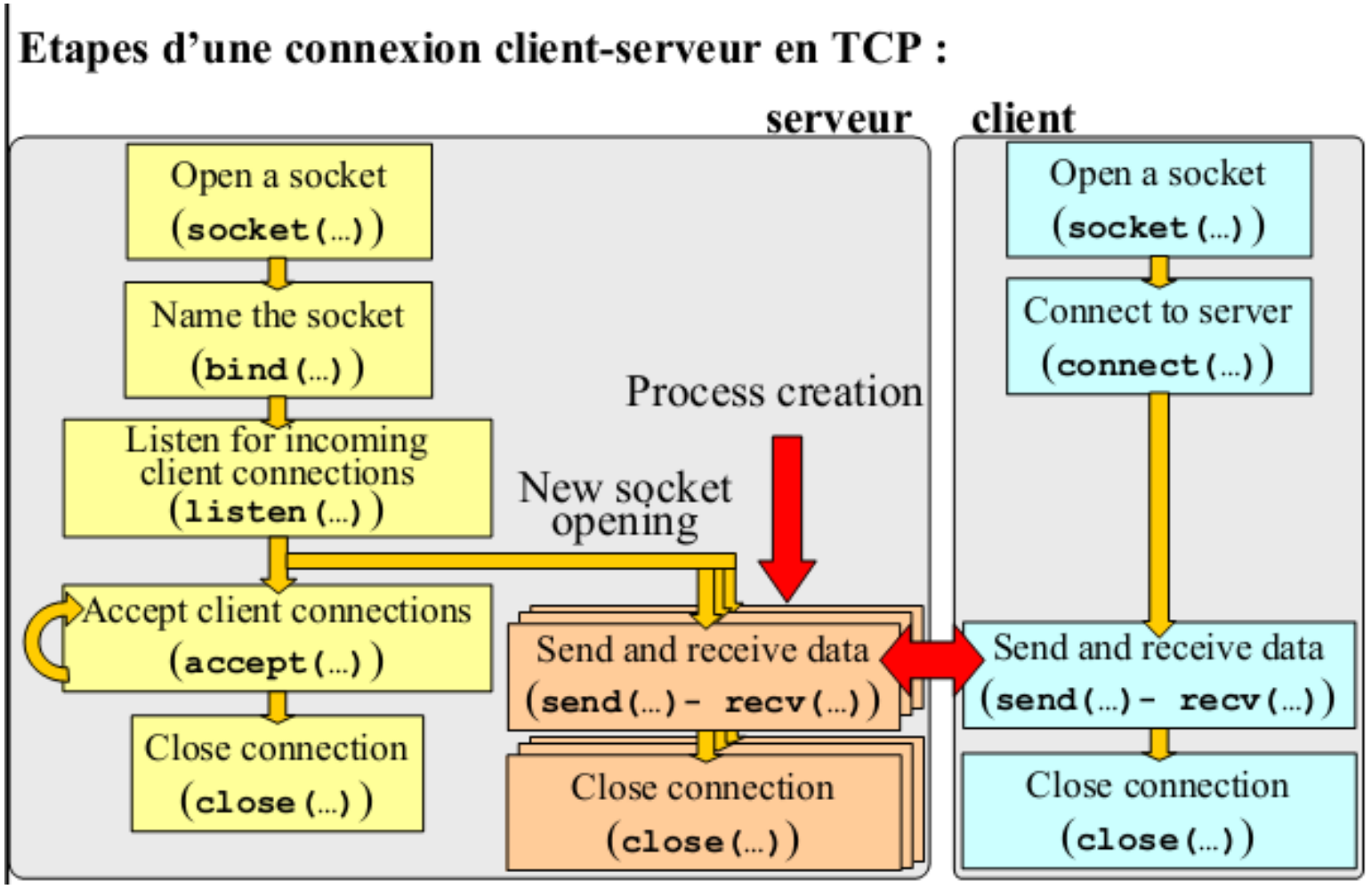
- Le serveur lie une socket d'écoute sur un certain port bien précis et appelle un service d'attente de connexion de la part d'un client
- Le client appelle un service pour ouvrir une connexion avec le serveur
- Il récupère une socket (associée à un port quelconque par le système)
- Du coté du serveur, le service d'attente de connexion retourne une socket de service (associée à un port quelconque)
- C'est la socket qui permet de dialoguer avec ce client
- le client et le serveur communiquent en envoyant et recevant des données via leur socket

Client-serveur en mode connecté

- Dans ce mode, on établit d'abord une connexion. On peut ensuite transférer des données à travers le tuyau créé.



Client-serveur en mode connecté



Client-serveur en mode connecté

Etapes d'une connexion client-serveur en TCP :

- le serveur et le client **ouvrent** chacun une « socket »
- le serveur la **nomme** (il l'attache à un de ses ports (un port précis))
- le client n'est pas obligé de la nommer (elle sera attachée automatiquement à un port lors de la connexion)
- le serveur **écoute** sa socket nommée
- le serveur **attend** des demandes de connexion
- le client **connecte** sa socket au serveur et à un de ses ports (précis)
- le serveur **détecte** la demande de connexion
- une nouvelle socket est ouverte automatiquement
- le serveur crée un processus pour **dialoguer** avec le client
- le nouveau processus continue le dialogue sur la nouvelle socket
- le serveur attendre en parallèle de nouvelles demandes de connexions
- finalement toutes les sockets doivent être **refermées**

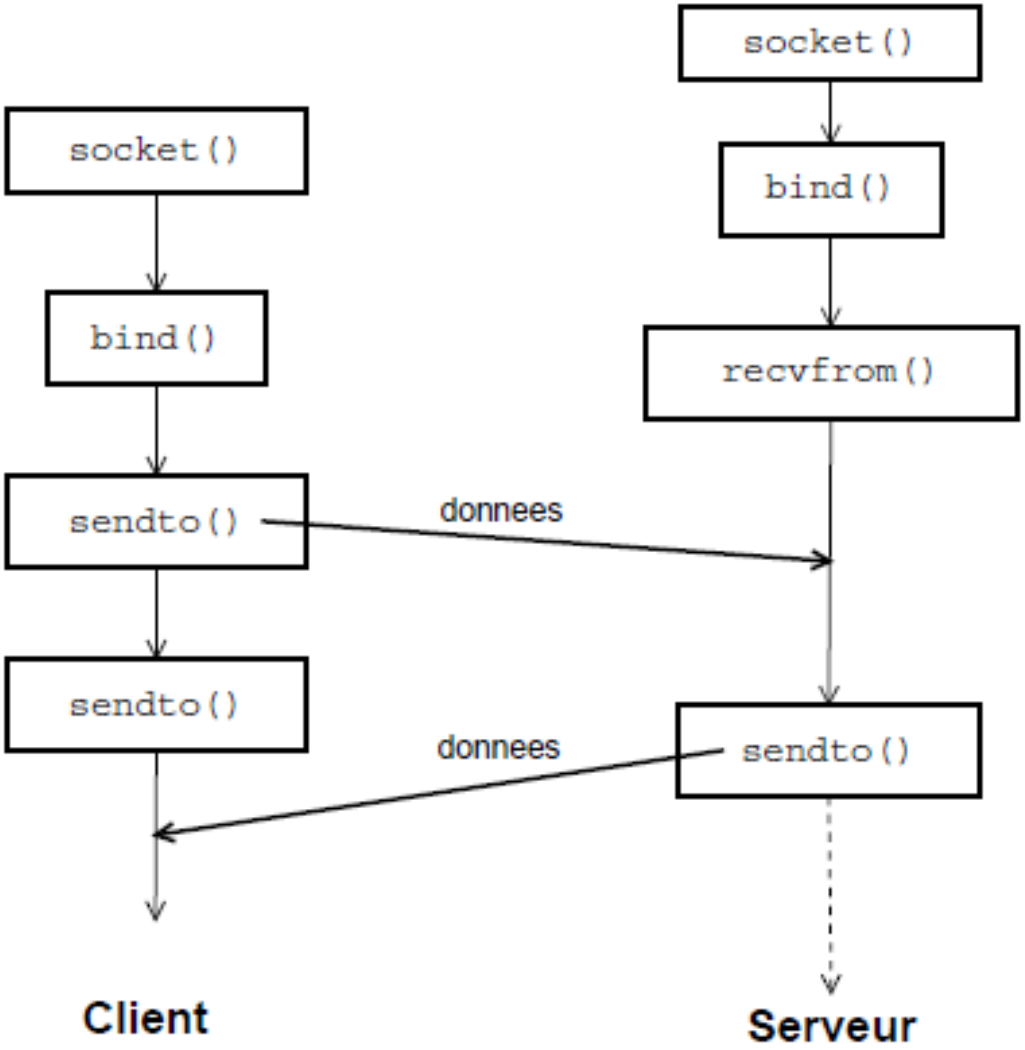
Utilisation des sockets en mode connecté SOCK_STREAM

- Côté Client (demandeur de la connexion) : le socket est dit actif
- crée un socket socket()
- se connecte à une <adresse,port> connect()
- lit et écrit dans le socket read(),recv();write(),send()
- ferme le socket close()

- Côté Serveur (en attente de connexion) : le socket est dit passif
- crée un socket socket()
- associe une adresse au socket bind()
- se met à l ’écoute des connexions entranteslisten()
- accepte une connexion entrante accept()
- lit et écrit sur le socket read(),recv();write(),send()
- ferme le socket close()

Client-serveur en mode non-connecté

- Dans ce mode, on n'établit pas de connexion au préalable.



Utilisation des sockets en mode non connecté SOCK_DGRAM

Côté Emetteur
- crée un socket socket()
- associe une adresse au socket bind()
- envoi d ’un message dans le socket sendto()/sendmsg()
- libère le socket close()

Côté Récepteur
- crée un socket socket()
- associe une adresse au socket bind()
- écoute et réception d ’un message recvfrom()/recvmsg()
- libère le socket close()

Transfert en mode non-connecté

- Aussi appelé *mode datagramme*.
- On utilise les commandes sendto() et recvfrom().

```
int
sendto(int s, const void *msg, size_t len, int flags
       const struct sockaddr *to, socklen_t tolen);
int
recvfrom(int  s,  void  *buf,  size_t len, int flags
         struct sockaddr *from, socklen_t *fromlen);
```

Obtention d'informations

- getpeername : retourne le nom de la machine connectée à une socket

Merci pour votre attention