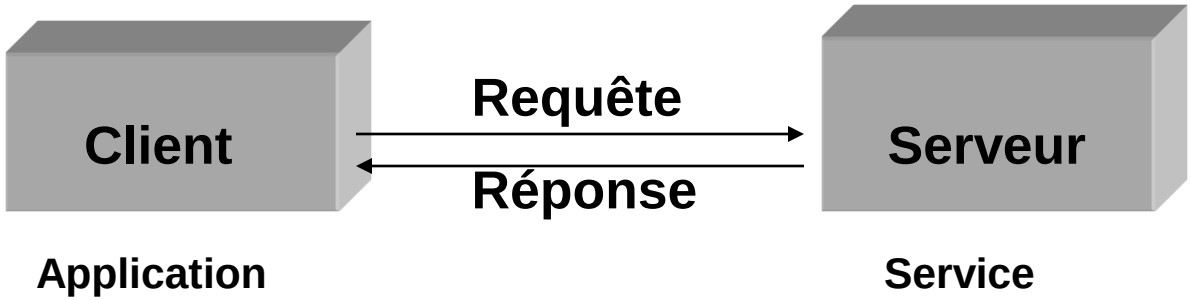
LE CLIENT/SERVEUR

- **Communication entre processus deux à deux: un client, un serveur;**
- **Ces processus forment un système coopératif:**
  - **le client réceptionne les résultats finaux délivrés par le serveur.**



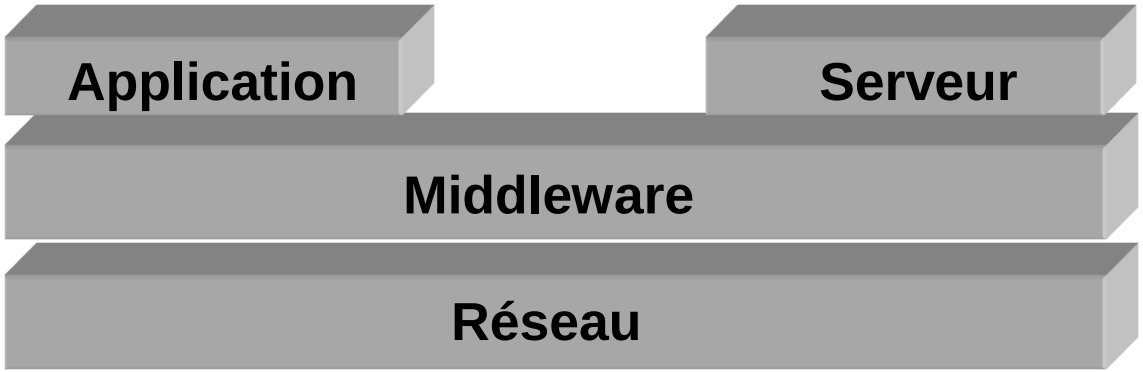
Modèle client/serveur ==> répartition des services plutôt que l'application elle-même.

UTILISATION DU MODÈLE C/S

- **S'étend de plus en plus vers tous les domaines d'activités :**
  - **Gestion des bases de données,**
  - **Systèmes transactionnels,**
  - **Systèmes de messagerie, Web, Intranet,**
  - **Systèmes de partage des données,**
  - **Calcul scientifique**
  - **...**

LE MIDDLEWARE

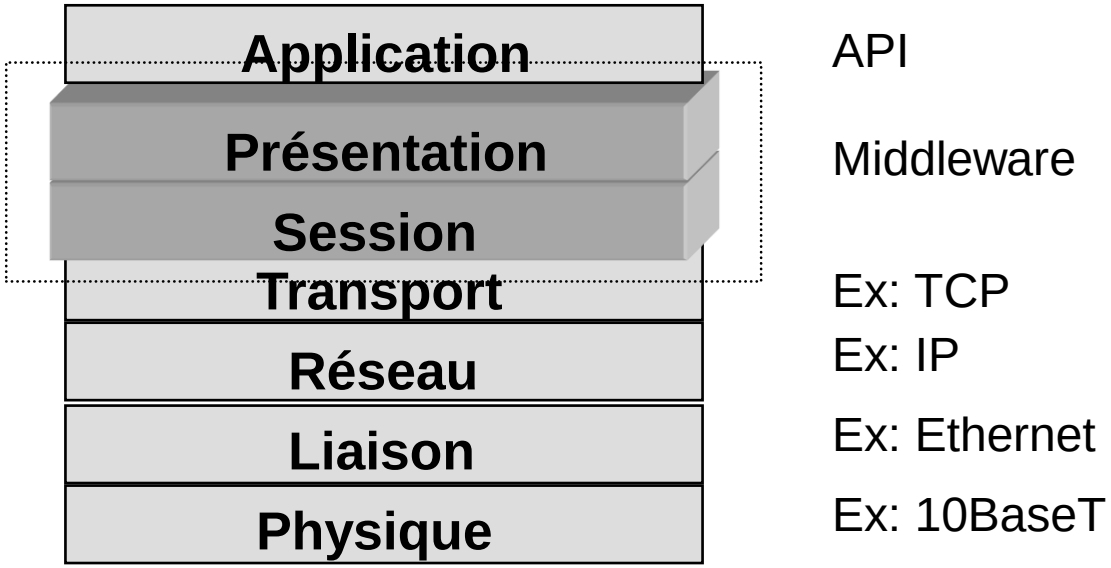
- **Complément de services du réseau permettant la réalisation du dialogue client/serveur :**
  - **prend en compte les requêtes de l'application cliente,**
  - **les transmet au serveur de manière transparente**
  - **prend en compte les données résultat du serveur vers le client.**



- **Le middleware offre aux applications une interface unifiée permettant l'accès à l'ensemble des services disponibles sur le réseau: l' API**

LE MIDDLEWARE

- **Fait le lien avec la couche Transport.**
- **Réalise la synchronisation du dialogue entre client et serveur,**
- **Définit le format des données échangées,**



COMPOSANTES DU C/S

- **On structure les applications en clients et serveurs en trois couches:**
  - **La couche présentation (interface Homme/Machine):**
    - Gestion de l'affichage (exemple Windows, X-windows, etc.),
    - Logique de l'affichage qui transmet les éléments de présentation.
  - **La couche traitements (ou logique):**
    - Algorithmique de l'application,
    - Gestion des traitements associés a a la manipulation des données de l'applicatif (ex: procédures SQL).
  - **La couche données pour la gestion des données** (SGDBD,…)

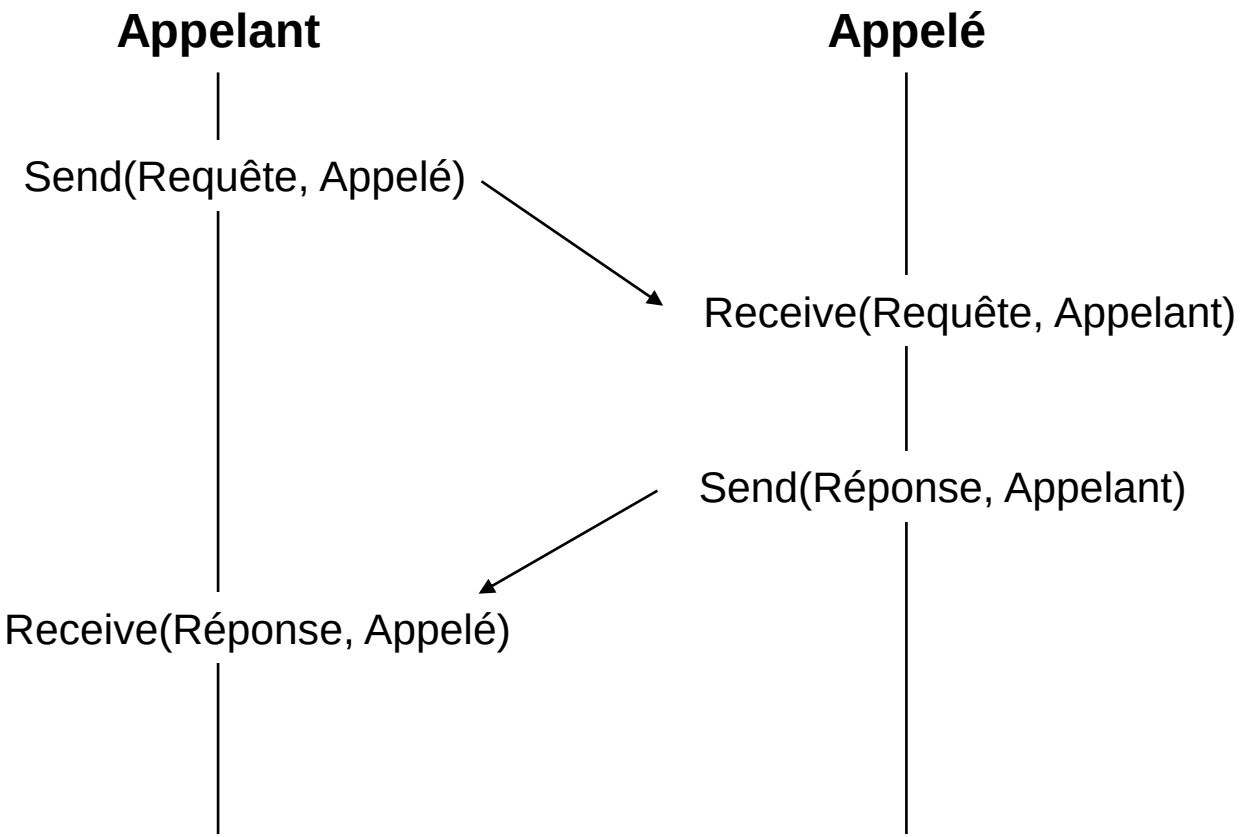
LES MODES DE CONNEXION

- **Le mode connecté:**
  - Une connexion est établie durant la session
  - Flot de données bidirectionnel
  - Quand la session est terminée, la connexion est fermée
  - Exemple: TCP

- **Le mode non connecté:**
  - Les messages sont envoyées de façon autonome et peuvent arriver dans un ordre quelconque
  - Exemple: IP , UDP
  - Un protocole connecté peut reposer sur un protocole non connecté (ex. TCP sur IP). L'ordonnancement, la non duplication, … doivent être gérés par l'application.
  - Un protocole non connecté peut reposer sur un protocole connecté (ex. HTTP sur TCP)

LES MODÈLES DE COMMUNICATION

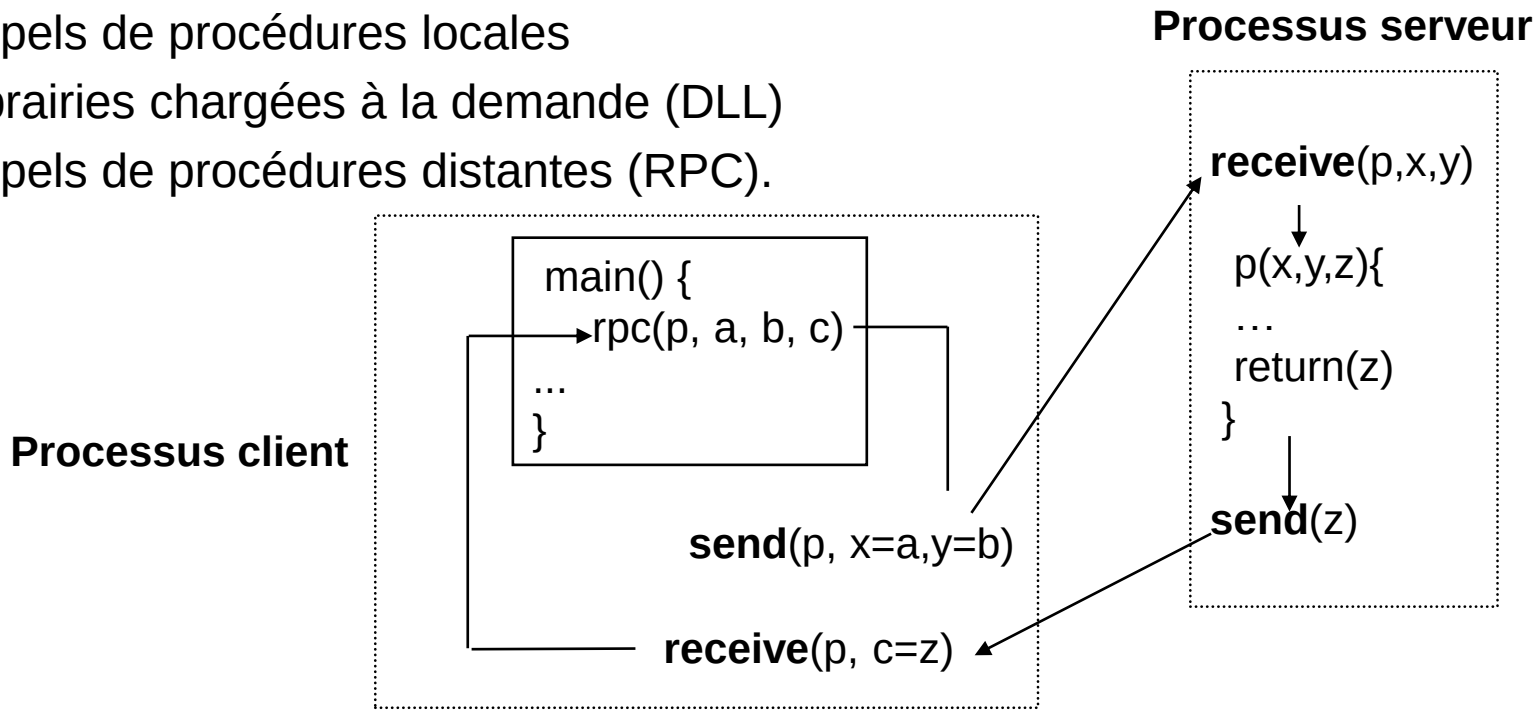
Envois de messages:



LES MODÈLES DE COMMUNICATION

- **Par évènements:**
  - Événements asynchrones (Windows, X-Windows,…)

- **Par appels de procédures:**
  - Appels de procédures locales
  - Librairies chargées à la demande (DLL)
  - Appels de procédures distantes (RPC).



LES CLIENTS

- **Une application cliente est moins complexe qu'une son 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. Alors que le serveur nécessite des accès privilégiés de connexion au middleware.

LES SERVEURS

- **Processus serveur:**
  - Offre une connexion sur le réseau,
  - Entre indéfiniment dans un processus d'attente de requêtes des clients,
  - Lorsqu'une requête arrive, le serveur déclenche les processus associés à cette requête, puis émet la ou les réponses vers le client.

- **Deux types de serveurs**
  - Itératifs: ne gèrent qu'un seul client à la fois
  - Parallèles : fonctionnent en mode concurrent.

QUEL TYPE DE SERVEUR ?

- **Serveurs itératifs en mode non-connecté :**
  - Services qui nécessitent très peu de traitement par requête (pas de concurrence). Exemple: serveur TIME

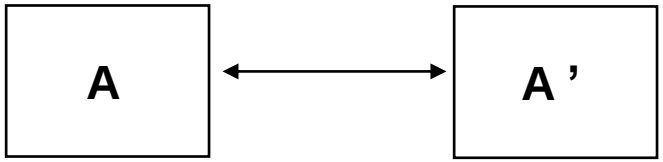
- **Serveurs itératifs en mode connecté :**
  - Services qui nécessitent très peu de traitement par requête mais requièrent un transport fiable de type TCP.

- **Serveurs concurrents en mode non-connecté :**
  - Temps de création d'un processus extrêmement faible par rapport au temps de traitement d'une requête,
  - Les requêtes nécessitent des accès périphériques importants.

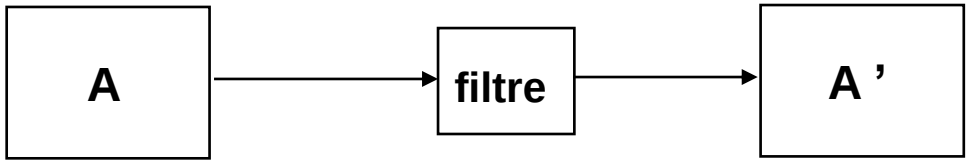
- **Serveurs concurrents en mode connecté :**
  - Les processus de service aident à compenser la lenteur du traitement imposés par TCP et réduisent les temps de réponse.

Modèles du Client/Serveur

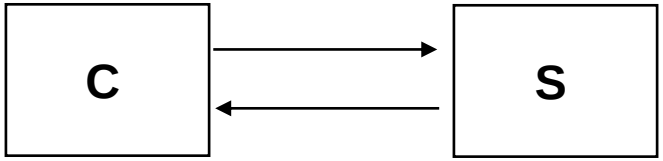
Peer-to-Peer:



Modèle avec filtre (Proxy):

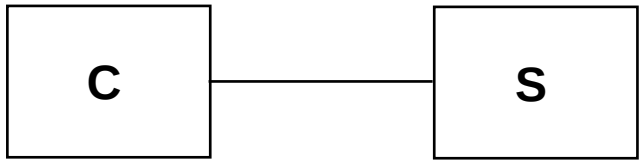


Client/Serveur:

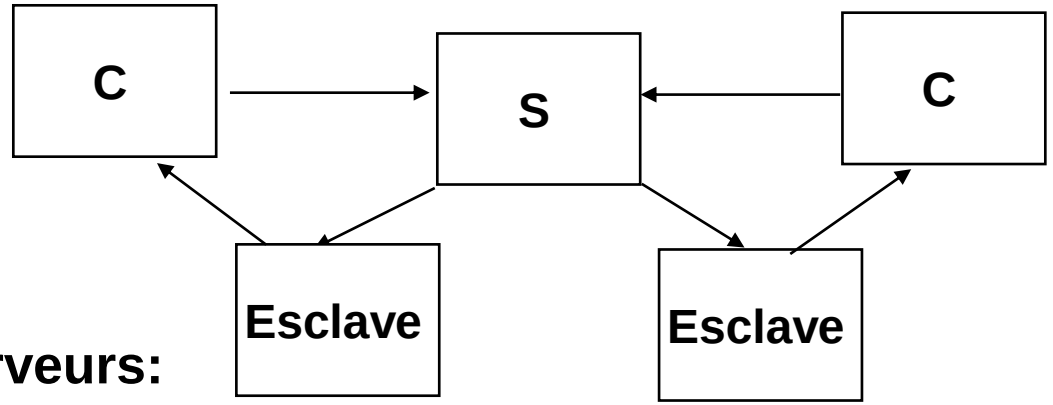


Distribution de services

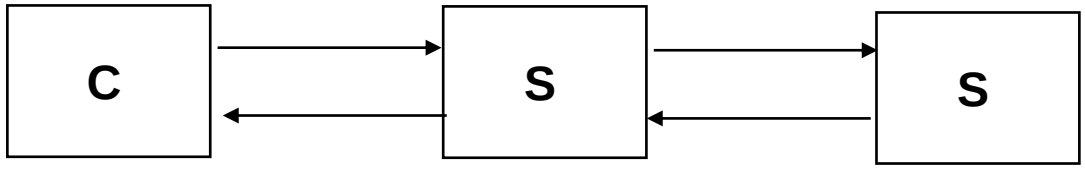
Un client- un serveur:



Plusieurs clients- un serveur:



Un client - plusieurs serveurs:

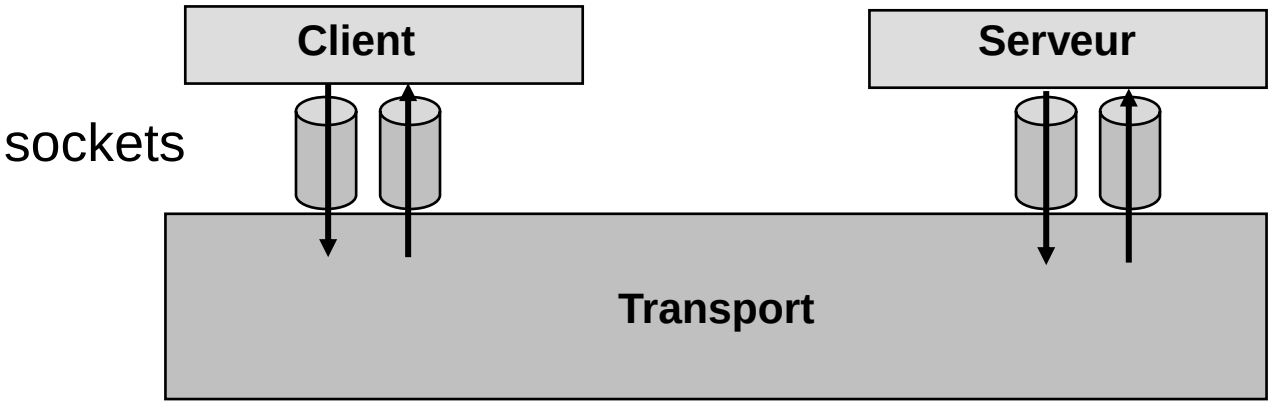


LES SOCKETS

- **Les sockets forment une interface client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP.**

- **Fournit les primitives pour le support des communications reposant sur les protocoles TCP/IP. Les applications cliente et serveur ne voient les couches de communication qu'à travers l'API socket.**

- **Une librairie d'appels de système qui permettent aux programmeurs d'accéder aux sockets.**



DOMAINES D'APPLICATIONS DES SOCKETS

- **On peut utiliser les sockets pour créer une bibliothèque d'utilitaires pour des programmes d'applications.**

- **Ces utilitaires fournissent les services suivants:**

  - **Lancer l'exécution d'un logiciel situé sur une machine distante (serveur);**

  - **Échanger de données entre l'application locale et le serveur, cela même si les représentations internes des deux machines sont différentes;**

  - **Dans des applications de bases de données, les sockets permettent de s'appuyer sur une machine puissante en lui associant des stations de travail qui prennent en charge l'interface utilisateur.**

STRUCTURE DES SOCKETS

- **Il existe différents types de sockets déterminées selon les propriétés de communication que l'on veut établir.**

- **Les propriétés des sockets sont:**

  - **Respect de la séquentialité des envois;**
  - **Efficacité de distribution;**
  - **Éviter les doubles envois;**
  - **Messages à caractère urgent;**
  - **Mode connecté;**
  - **Socket nommée.**

CARACTERISTIQUES DES SOCKETS

- **Un socket possède trois caractéristiques:**
  - **Type: Décrit comment 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
  - **Protocole: Décrit les protocoles de communication utilisés pour émettre et recevoir des données:** SOCK_DGRAM TCP, SOCK_STREAM UDP, ...

STRUCTURES D ’ADRESSES

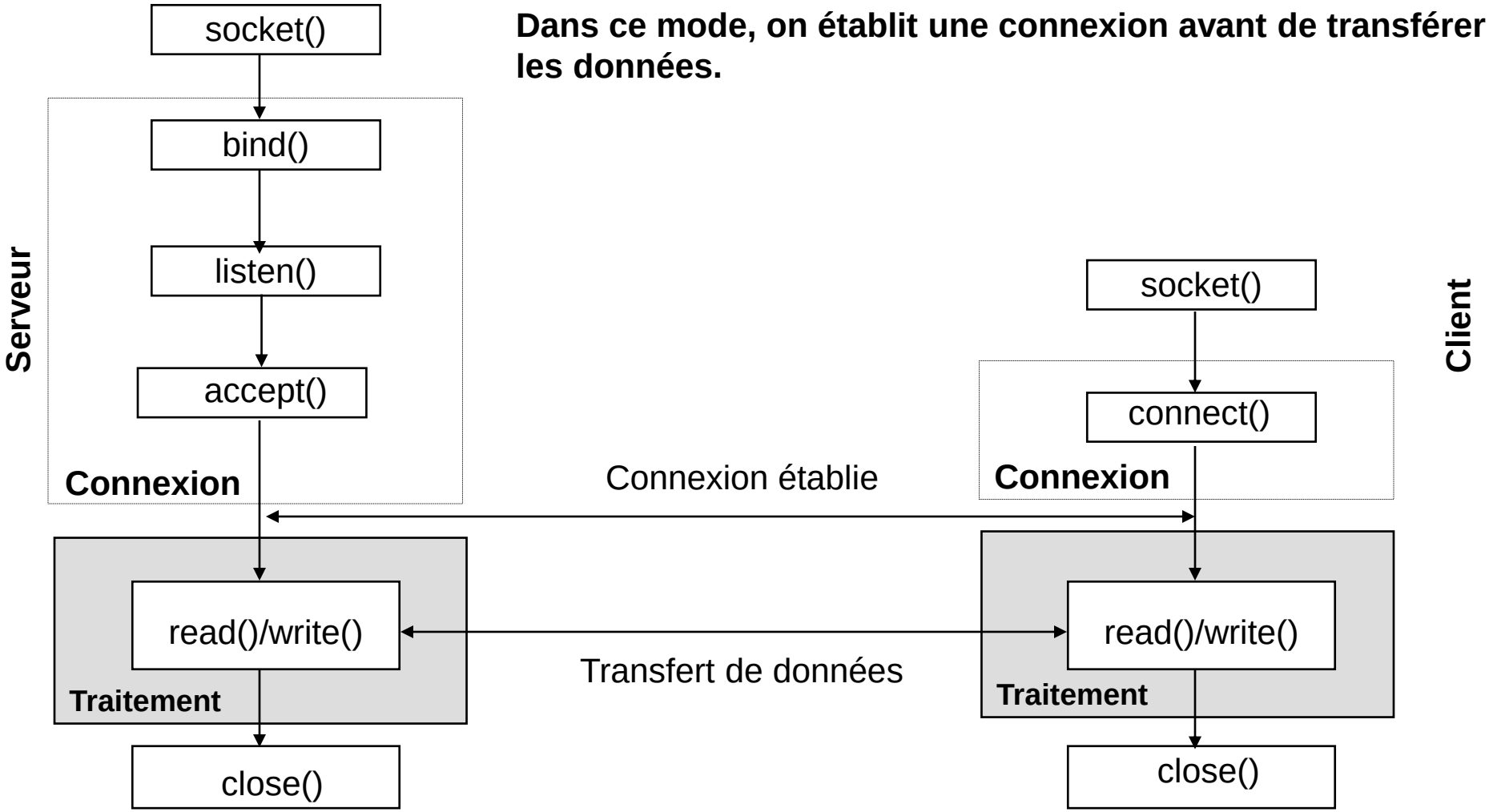
- **Une socket utilise une structure d ’adresses générique adaptable à la famille d ’adresses sélectionnée par l ’application:**

```
struct sockaddr {
        u_short sa_family ;
        char sa_data[14];
}

struct sockaddrd_in {
        u_short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];
}
```

CLIENT-SERVEUR EN MODE CONNECTÉ

Dans ce mode, on établit une connexion avant de transférer les données.



CLIENT EN MODE CONNECTÉ

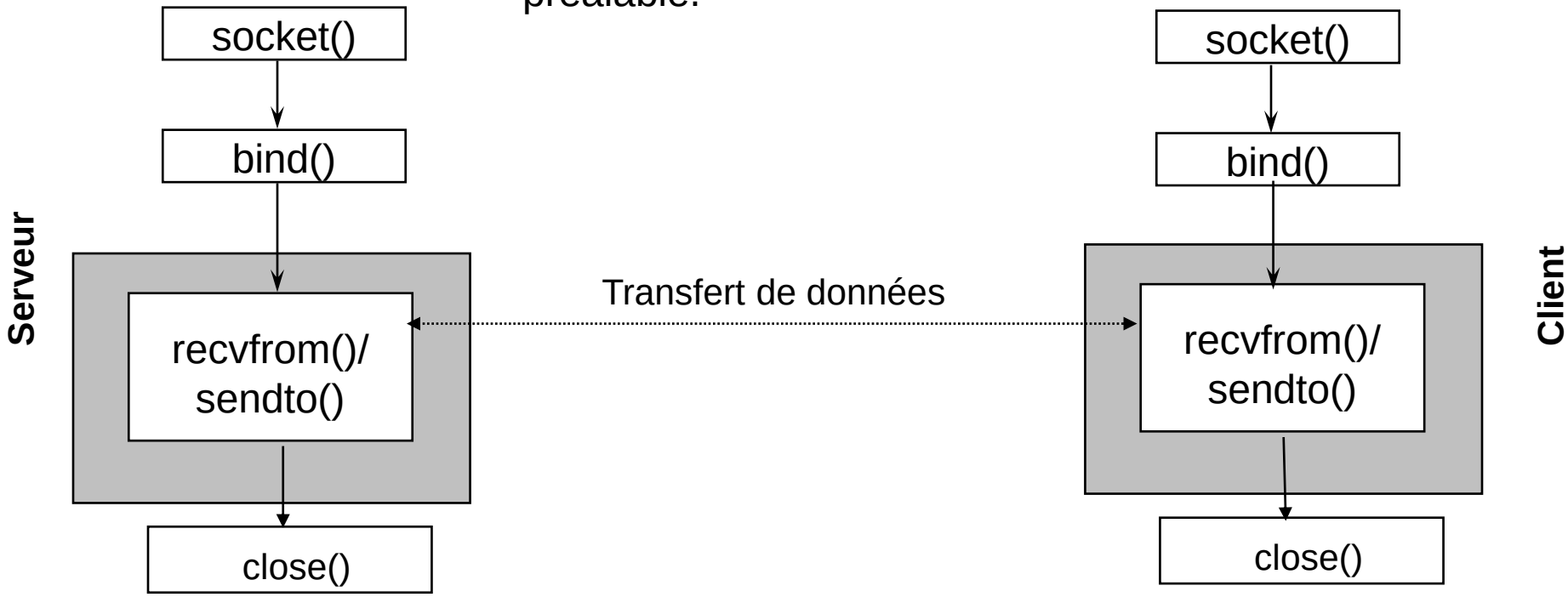
- **On suit les étapes suivantes:**
  - **Créer une socket,**
  - **Se connecter au serveur en donnant l'adresse socket distante (Adresse Internet du serveur et numéro de port du service). Cette connexion attribue automatiquement un numéro de port au client,**
  - **Lecture ou écriture sur la socket.**

SERVEUR EN MODE CONNECTÉ

- **On suit les étapes suivantes:**
  - **Créer une socket,**
  - **Associer une adresse socket (Adresse Internet et numéro de port) au service,**
  - **Se mettre à l'écoute des connexions entrantes,**
  - **Pour chaque connexion:**
    - **Accepter la connexion (une nouvelle socket dérivée de la principale est créée);**
    - **Lire ou écrire sur la nouvelle socket;**
    - **Fermer la nouvelle socket.**

CLIENT-SERVEUR EN MODE NON CONNECTÉ

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



SOCKET EN MODE NON CONNECTÉ

- **Étapes à suivre:**
  - **Client:**

    Créer une socket,

    Associer une adresse (Adresse Internet et numéro de port) au service,

    Lecture ou écriture sur la socket.

  - **Serveur:**

    Créer une socket,

    Associer une adresse socket au service,

    Lecture ou écriture sur la socket.

PRIMITIVES DES SOCKETS

- **La première étape consiste à ouvrir un socket avec la primitive** socket()**.**
- **La primitive** bind() **permet d'associer une adresse à un socket.**
- **La commande** connect() **est utilisée par le client pour établir la connexion TCP lorsque le domaine est** AF_INET **et le Type est** STREAM **ou identifier le serveur lorsque le domaine est** AF_INET **et le Type est** DTGRAM.
- **La commande** accept() **est utilisée par le serveur pour établir la connexion TCP lorsque le domaine est** AF_INET **et le type est** STREAM**.**
- **La commande** listen() **est utilisée par le serveur pour établir la connexion TCP.**

PRIMITIVES DES SOCKETS

- **La commande** send() **permet d'envoyer des données. Elle permet de choisir certains paramètres. Dans le champs flag on peut spécifier** MSG_OOB **pour expédier des données urgentes. On peut aussi utiliser la commande** write().

- **La commande** sendto() **est comme** send() **mais en mode non connecté.**

- **La commande** recv() **permet de recevoir des données. Elle permet de choisir le flag** MSG_OOB **pour expédier des données urgentes et le flag** MSG_PEEK **pour garder les données lues dans le socket**. **On peut aussi utiliser la commande** read().

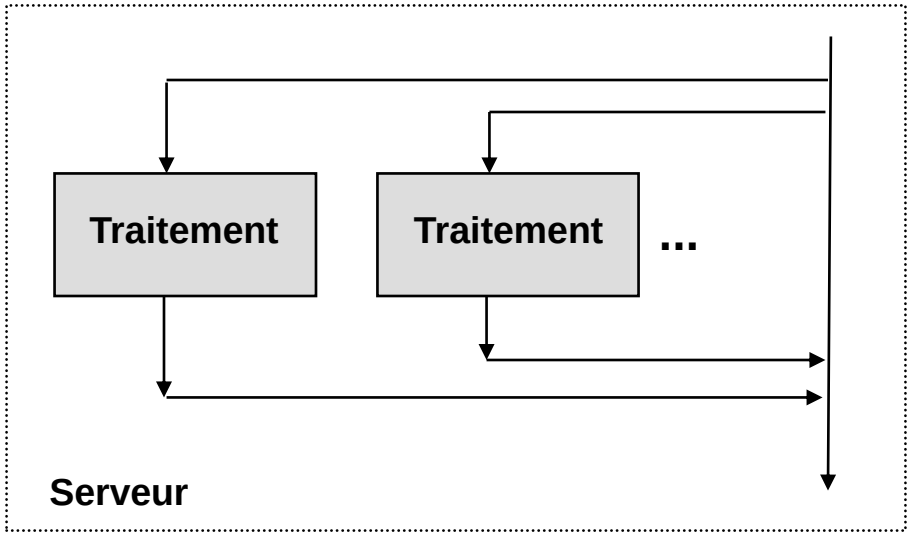
- **La commande** recvfrom() **permet de recevoir des données en mode non connecté.**

- **La commande** shutdown() **(ou** close()**) permettent de fermer une socket. Elle permet également d 'indiquer le mode de fermeture (i.e.:** 0: plus de lecture, 1: plus d'écriture, 2: plus de lecture-écriture).

SERVEUR PARALLELE

Le serveur sert plusieurs connexions en parallèle:

```
sd=socket(...);
bind(...);
listen(sd,5);
for( ; ; ) {
 nsd = accept(sd,…);
 if(fork() == 0) {
            close(sd);
            Traitement(nsd);
            exit(0);
}
else {
    close(nsd);
    ...
}
```



SERVEUR ITERATIF

Le serveur sert une connexion à la fois:

```
sd=socket(...);
bind(...);
listen(sd,5);
for( ; ; ) {
   nsd = accept(sd,…);
   Traitement(nsd);
   close(nsd);
    ...
}
```