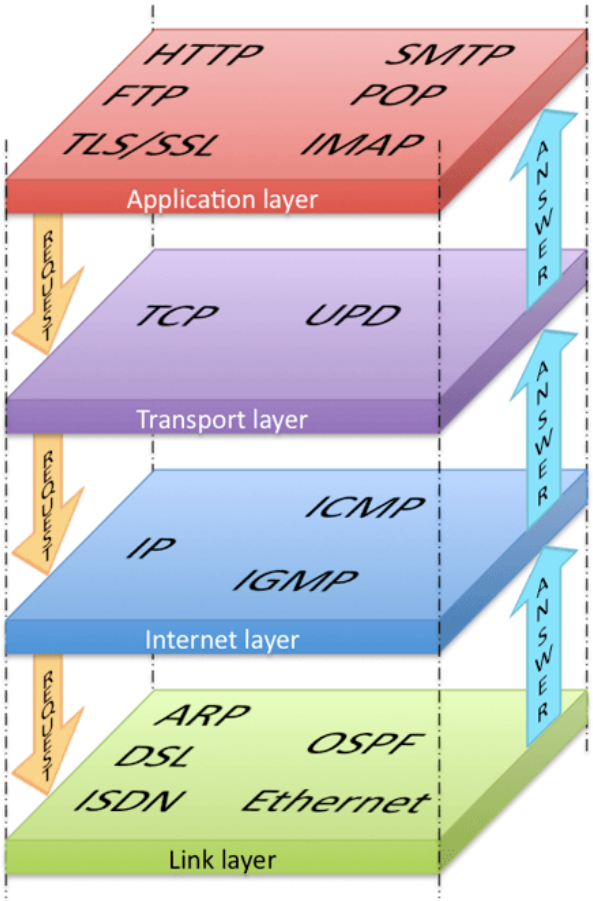
Sockets TCP/IP

Couche Transport et Application

Hinde Bouziane

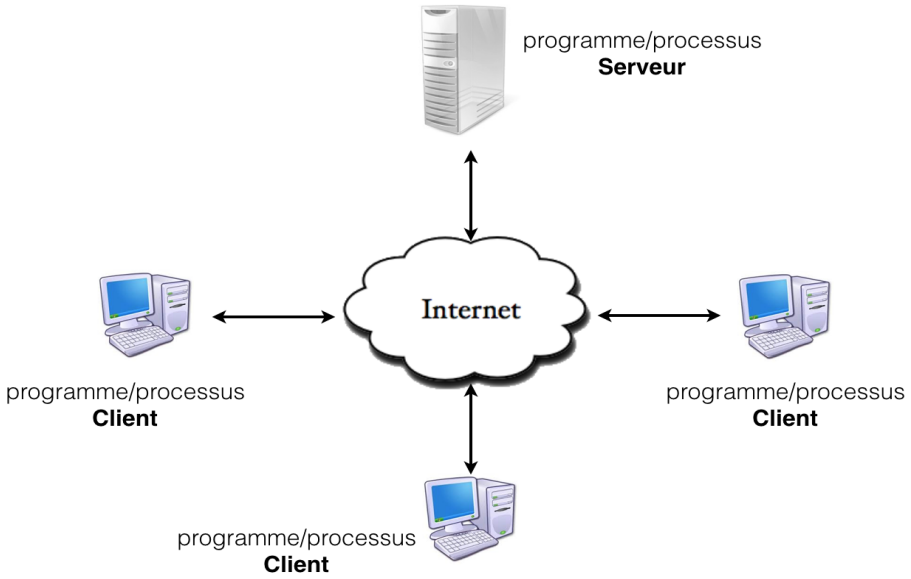
bouziane@lirmm.fr

Positionnement OSI : Vision en 4 couches



Objectif

Concevoir et programmer des applications nécessitant des communications distantes en utilisant le protocole TCP/IP.



Plus précisément :

- les sockets et le protocole de transport TCP
- le protocole réseaux Internet
- le langage C - Noyau Unix

Le modèle client-serveur

Serveur :

processus qui attend des requêtes provenant de processus clients, réalise ces requêtes et rend (ou pas) les résultats.

Le modèle client-serveur

Serveur :

processus qui attend des requêtes provenant de processus clients, réalise ces requêtes et rend (ou pas) les résultats.

Client :

processus qui envoie des requêtes au processus dit *serveur*, attend une réponse (ou pas).

Le modèle client-serveur

Serveur :

processus qui attend des requêtes provenant de processus clients, réalise ces requêtes et rend (ou pas) les résultats.

Client :

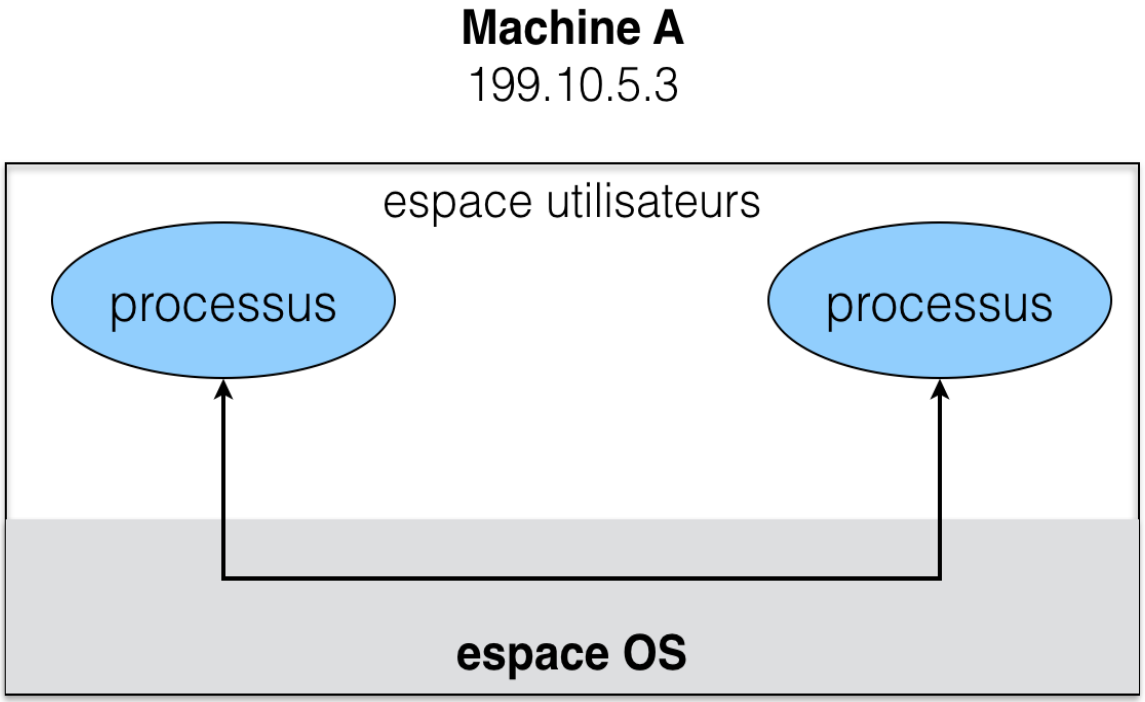
processus qui envoie des requêtes au processus dit *serveur*, attend une réponse (ou pas).

Requête / réponse :

suite d'instructions, commandes, simple chaîne de caractères, etc. obéissant à un langage, un accord ou une structure préalables connus des deux entités communicantes (protocole d'application).

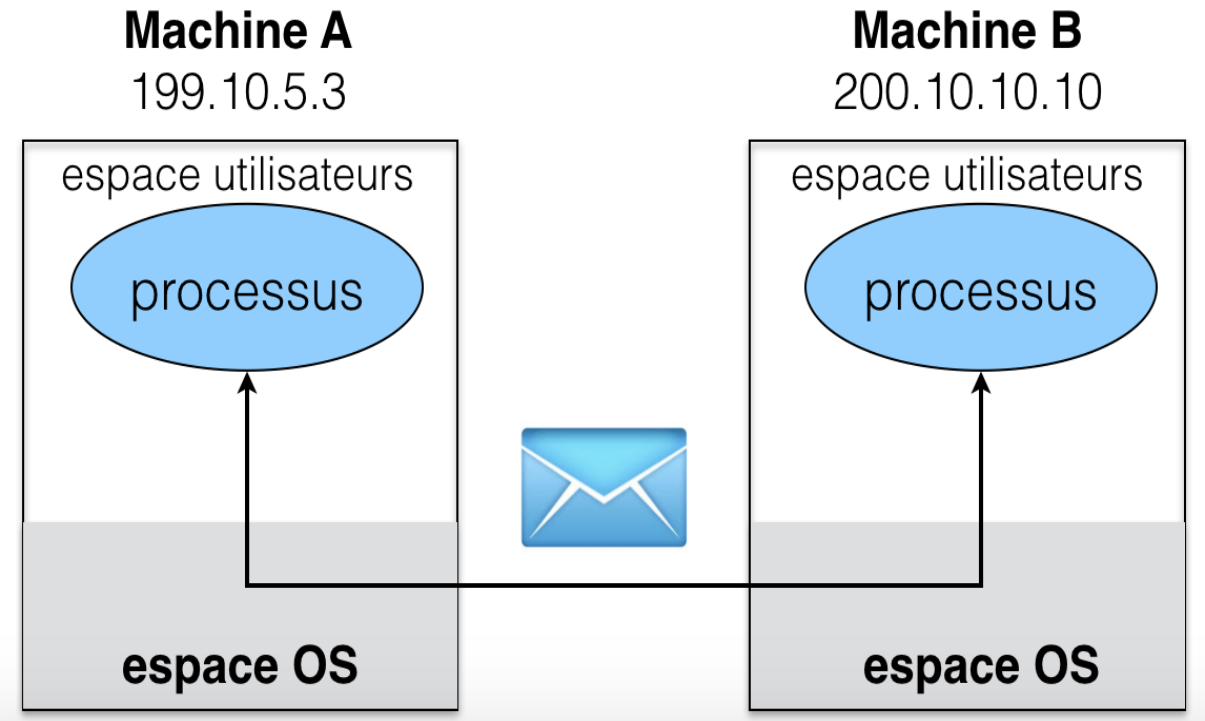
Contenu

Communications dans les systèmes centralisés



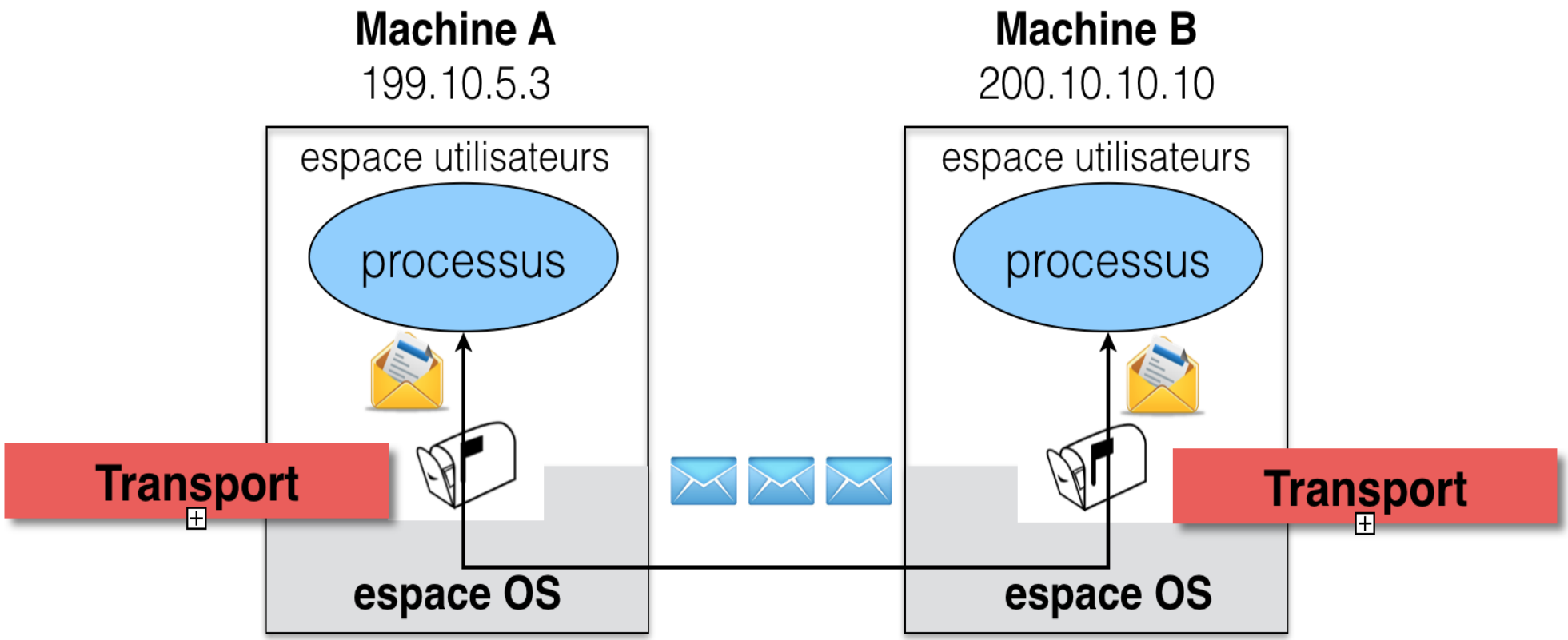
Pipes/tubes, IPC
Supposent l'existence d'une mémoire partagée entre processus

Communications dans les systèmes répartis / distribués



Envoi / réception de messages

Communications dans les systèmes répartis / distribués



± Socket + API

Qu'est ce qu'une socket ?

Définition 1

Une socket (en français, prise) est une notion qui étend celle de tube / pipe. Elle permet de définir un canal de communication entre deux processus, sauf qu'elle permet en plus :

- la communication distante (en réseaux) ;
- le choix de différents protocoles de communication.

Qu'est ce qu'une socket ?

Définition 1

Une socket (en français, prise) est une notion qui étend celle de tube / pipe. Elle permet de définir un canal de communication entre deux processus, sauf qu'elle permet en plus :

- la communication distante (en réseaux) ;
- le choix de différents protocoles de communication.

Définition 2

Une socket est une extrémité d'un canal de communication bidirectionnel entre deux processus

Qu'est ce qu'une socket ?

Définition 1

Une socket (en français, prise) est une notion qui étend celle de tube / pipe. Elle permet de définir un canal de communication entre deux processus, sauf qu'elle permet en plus :

- la communication distante (en réseaux) ;
- le choix de différents protocoles de communication.

Définition 2

Une socket est une extrémité d'un canal de communication bidirectionnel entre deux processus

Bidirectionnel

Dans les deux sens : envoi et réception.

Propriétés d'une socket

Concretement, une socket est définie par :

- un domaine d'appartenance
- un type
- un protocole
- une paire (adresse IP et numéro de port) pour la désigner dans le domaine.

Autres propriétés :

- une socket est identifiée par un descripteur de fichier
- une socket est associée à deux buffers :
  - de réception : contient les données reçues par la couche transport et à lire par la couche application,
  - d'émission : contient les données que la couche application transmet à la couche transport.
- Une socket est un concept multiplateforme et multi-langage.

Domaine de communication

Le domaine permet d'identifier une socket dans un domaine d'adresses et son utilisation dans une famille de protocoles connus.
Exemples :

- IPv4 (PF_INET) : L'adresse de la socket est une adresse IPv4 et la communication se fait avec un ou des processus distants suivant les protocoles Internet v4.
- IPv6 (PF_INET6) : L'adresse de la socket est une adresse IPv6 et la communication se fait avec un ou des processus distants suivant les protocoles Internet v6.
- Etc. : voir la documentation

Nous utiliserons le domaine IPv4 (PF_INET).

Types d'une socket (1/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Types d'une socket (1/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Deux formats : datagramme et stream

- Datagramme (SOCK_DGRAM) : un message est expédié comme un paquet bien délimité et est reçu entièrement en une seule fois.
- Stream (SOCK_STREAM) : un message est expédié/reçu comme un flot continu de caractères. Si l'expéditeur envoie plusieurs messages, le destinataire pourra lire ces messages en une fois, en plusieurs fois ou caractère par caractère. Les limites des messages ne sont pas définis.

Types d'une socket (1/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Deux formats : datagramme et stream

- Datagramme (SOCK_DGRAM) : un message est expédié comme un paquet bien délimité et est reçu entièrement en une seule fois.
- Stream (SOCK_STREAM) : un message est expédié/reçu comme un flot continu de caractères. Si l'expéditeur envoie plusieurs messages, le destinataire pourra lire ces messages en une fois, en plusieurs fois ou caractère par caractère. Les limites des messages ne sont pas définis.

Question : Dans le cas d'une socket SOCK_STREAM, à qui revient la responsabilité de définir les limites des messages ?

Types d'une socket (2/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Deux modes de communication : connecté et non connecté

- Connecté : la transmission de messages est précédée par une phase de connexion avec une autre socket. Une socket en mode connecté est donc utilisée pour communiquer de façon exclusive avec une seule autre socket. Nous parlons de *circuit ou canal virtuel* établi entre les deux sockets (analogie : communications téléphoniques)
- Non connecté : la destination d'un message à émettre via une socket en mode non connectée n'est pas nécessairement la même que celle du message suivant. A chaque émission, une adresse de destination doit être spécifiée (analogie : communications par courrier postal)

Types d'une socket (3/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Propriétés des transmissions

- Fiabilité : *soit un message transmis arrive bien à destination, soit une erreur est retournée à l'application.*
- Paquets/messages non dupliqués
- Remise dans l'ordre des paquets : *si des paquets sont reçus dans le désordre, la couche transport prendrait en charge la remise en ordre avant le dépôt d'un message à l'application.*

Types d'une socket (3/3)

Le type d'une socket détermine le format de transmission de données, le mode de connexion utilisé avec un ou d'autres processus et des propriétés de transmission offertes par la couche transport.

Propriétés des transmissions

- Fiabilité : *soit un message transmis arrive bien à destination, soit une erreur est retournée à l'application.*
- Paquets/messages non dupliqués
- Remise dans l'ordre des paquets : *si des paquets sont reçus dans le désordre, la couche transport prendrait en charge la remise en ordre avant le dépôt d'un message à l'application.*

Remarque : la garantie de ces propriétés dépend des protocoles utilisés.

Protocoles

Par défaut :

- Une socket de type datagramme (SOCK_DGRAM) utilise le protocole UDP (User Datagram Protocol)
- Une socket de type stream (SOCK_STREAM) utilise le protocole TCP (Transmission Control Protocol)

TCP	UDP
fiable	non fiable
ordre garanti	ordre non garanti
duplication impossible	duplication possible
mode connecté	mode non connecté

Il existe d'autres protocoles que vous pouvez trouver dans la documentation.

Désigner une socket dans un domaine : Nommage

- Pour pouvoir se connecter ou envoyer des données à une socket distante, il est nécessaire de connaitre son adresse (adresse IP, numéro de port).
- L'association d'une adresse à une socket est appelée : nommage.
- Deux méthodes de nommage :
  - Le programmeur ou l'utilisateur choisi une adresse IP et un numéro de port. Dans ce cas, le système vérifie leur disponibilité.
  - Laisser le système choisir une ou des adresses. Il est possible ensuite de consulter ces informations pour les connaitre.

Allocation des numéros de port

- Dans l'Internet, chaque application client-serveur va se voir attribuer un ou des numéros de port **publique(s)**.
- Les numéros jusqu'à 1024 sont officiellement réservés pour des applications connues et ne peuvent être demandés par une application d'utilisateur non administrateur.
- Exemples :
  - Tous les serveurs `sshd` utilisent strictement le port numéro 22 ;
  - Tous les serveurs `httpd` utilisent *par défaut* le port numéro 80.
- Pour vos programmes, les numéros de ports seront supérieurs à 1024.

En pratique ?

Utilisation d'une API pour :

- créer une socket
- nommer une socket
- se connecter à une socket distante
- recevoir un message
- envoyer un message
- fermer une socket

La connexion de deux sockets ainsi que la réception et l'envoi de messages seront présentés uniquement pour TCP

Création d'une socket

```
int socket(int domain, int type, int protocole)
```

Création d'une socket

```
int socket(int domain, int type, int protocole)
```

Exemple

int dSock = socket(PF_INET, SOCK_STREAM, 0) ;
Que fait cet appel ?

Création d'une socket

int socket(int domain, int type, int protocole)

Exemple

int dSock = socket(PF_INET, SOCK_STREAM, 0) ;
Que fait cet appel ?

Valeur de retour

Le descripteur de la socket créée ou -1 en cas d'erreur.

Nommage d'une socket (1/3)

```
int bind (int descripteur,                    // descripteur de socket
          const struct sockaddr *adresse,     // pointeur vers l'adresse
          socklen_t lgAdr)                    // longueur de l'adresse
```

Nommage d'une socket (1/3)

```
int bind (int descripteur,                    // descripteur de socket
          const struct sockaddr *adresse,     // pointeur vers l'adresse
          socklen_t lgAdr)                    // longueur de l'adresse
```

La structure sockaddr

```
struct sockaddr {
    sa_family_t sa_family;   // famille d'adresses, AF_xxx
    char sa_data[14];   // 14 octets pour l'IP + port
};
```

Elle définit un type générique d'adresses. Le type réellement utilisé varie en fonction du domaine de la socket :

- Si PF_INET *alors struct sockaddr_in : une adresse IPv4 et un port*
- Si PF_INET6 alors struct sockaddr_in6 : une adresse IPv6 et un port

Nommage d'une socket (2/3)

La structure sockaddr_in

```
struct sockaddr_in {
    sa_family_t sin_family;    // famille AF_INET
    in_port_t sin_port;    // numéro de port au format réseau
    struct in_addr sin_addr;    // structure d'adresse IP
};
struct in_addr {    uint32_t s_addr;  // adresse IP au format réseau };
```

Remarques

- L'IP et le numéro de port sont stockés au format réseau (network byte order) : XXxx
- Les entiers sont au format hôte (host byte order) : xxXX ou XXxx
- Une conversion est nécessaire : fonctions ntohs(), htons(), ntohl(), htonl()

Nommage d'une socket (3/3)

Exemple : que fait le code suivant ?

```
int dSock = socket(PF_INET, SOCK_STREAM, 0);
struct sockaddr_in ad;
ad.sin_family = AF_INET;
ad.sin_addr.s_addr = INADDR_ANY;
ad.sin_port = htons((short)31470);
int res = bind(dSock, (struct sockaddr*)&ad, sizeof(ad));
```

INADDR_ANY

Attache la socket à toutes les interfaces réseaux locales.

Valeur de retour

0 si le nommage a réussi, -1 sinon (e.g. si le port est déjà utilisé).

Fermeture d'une socket

```
int close (int descripteur) ;
int shutdown(int descripteur, int comment) ;
                        // comment : SHUT_WR arrêt émission ou
                        // SHUT_RDWR pour l'émission et la réception
```

Valeur de retour

0 si la fermeture a réussi, -1 sinon.

Principe (illustré avec deux programmes)

Client **Serveur**

Principe (illustré avec deux programmes)

Client

Créer une socket

Serveur

Créer une socket

Principe (illustré avec deux programmes)

Client

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Serveur

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;

Principe (illustré avec deux programmes)

Client

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Serveur

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Nommer la socket

bind(dS, ...) ;

Passer la socket en mode écoute

Principe (illustré avec deux programmes)

Client

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Serveur

Créer une socket

dS = socket(.., SOCK_STREAM, ..) ;

Nommer la socket

bind(dS, ...) ;

Passer la socket en mode écoute

listen(dS, ...) ;

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;
Communiquer	Communiquer

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;
Communiquer	Communiquer
send(dS, ...) ;	recv(dSClient, ...) ;

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;
Communiquer	Communiquer
send(dS, ...) ;	recv(dSClient, ...) ;
recv(dS, ...) ;	send(dSClient, ...) ;
...	...

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;
Communiquer	Communiquer
send(dS, ...) ;	recv(dSClient, ...) ;
recv(dS, ...) ;	send(dSClient, ...) ;
...	...
Fermer la Socket	Fermer les sockets

Principe (illustré avec deux programmes)

Client	**Serveur**
Créer une socket	Créer une socket
dS = socket(.., SOCK_STREAM, ..) ;	dS = socket(.., SOCK_STREAM, ..) ;
	Nommer la socket
	bind(dS, ...) ;
	Passer la socket en mode écoute
	listen(dS, ...) ;
Demander une connexion	Accepter une connexion
connect(dS, ...) ;	dSClient = accept(dS, ...) ;
Communiquer	Communiquer
send(dS, ...) ;	recv(dSClient, ...) ;
recv(dS, ...) ;	send(dSClient, ...) ;
...	...
Fermer la Socket	Fermer les sockets
close(dS) ;	close(dSClient) ; close(dS) ;

Passer une socket en mode écoute

```
int listen(int descripteur, int nbMaxEnAttente)
```

Passer une socket en mode écoute

```
int listen(int descripteur, int nbMaxEnAttente)
```

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
int res = listen(dS, 10);
```

Passe la socket dont le descripteur est dS en écoute de demandes de connexion. 10 est le nombre maximum de demandes de connexion pouvant être mises en attente (fixe une longueur de file d'attente).

Passer une socket en mode écoute

```
int listen(int descripteur, int nbMaxEnAttente)
```

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
int res = listen(dS, 10);
```

Passe la socket dont le descripteur est dS en écoute de demandes de connexion. 10 est le nombre maximum de demandes de connexion pouvant être mises en attente (fixe une longueur de file d'attente).

Valeur de retour

0 en cas de succès, -1 sinon (avec errno positionné).

Demande de connexion à un serveur

```
int connect(int descr, const struct sockaddr *adServ, socklen_t lgAdr)
```

Demande de connexion à un serveur

int connect(int descr, const struct sockaddr *adServ, socklen_t lgAdr)

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
struct sockaddr_in adServ ;
adServ.sin_family = AF_INET ; adServ.sin_port = htons(34567) ;
int res = inet_pton(AF_INET, "197.50.51.10", &(adServ.sin_addr)) ;
socklen_t lgA = sizeof(struct sockaddr_in) ;
res = connect(dS, (struct sockaddr *) &adServ, lgA) ;
```

Envoie une demande de connexion de la socket du client dS au serveur, via la socket du serveur dont l'IP est 197.50.51.10 et le numéro de port 34567.

Note : *favorisez le passage de paramètres ou la saisie des IP et numéro de port.*

Demande de connexion à un serveur

int connect(int descr, const struct sockaddr *adServ, socklen_t lgAdr)

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
struct sockaddr_in adServ ;
adServ.sin_family = AF_INET ; adServ.sin_port = htons(34567) ;
int res = inet_pton(AF_INET, "197.50.51.10", &(adServ.sin_addr)) ;
socklen_t lgA = sizeof(struct sockaddr_in) ;
res = connect(dS, (struct sockaddr *) &adServ, lgA) ;
```

Envoie une demande de connexion de la socket du client dS au serveur, via la socket du serveur dont l'IP est 197.50.51.10 et le numéro de port 34567.

Note : *favorisez le passage de paramètres ou la saisie des IP et numéro de port.*

Valeur de retour

0 en cas de succès, -1 sinon (avec errno positionné).

Accepter une demande de connexion d'un client

```
int accept(int descr,                  /* descripteur de la socket recevant des
                                          demandes de connexion*/
    struct sockaddr *adrClient,        // pointeur vers l'adresse de la socket du client
    socklen_t lgAdr)                   // pointeur vers longueur de l'adresse
```

Accepter une demande de connexion d'un client

```
int accept(int descr,                /* descripteur de la socket recevant des
                                        demandes de connexion*/
    struct sockaddr *adrClient,      // pointeur vers l'adresse de la socket du client
    socklen_t lgAdr)                 // pointeur vers longueur de l'adresse
```

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
int res = listen(dS, 10);
struct sockaddr_in adClient; socklen_t lgA = sizeof(struct sockaddr_in);
int dSClient = accept(dS, (struct sockaddr *) &adClient, &lgA);
```

Extrait une demande de connexion de la socket dS et la traite. En cas de succès, une nouvelle socket (descripteur dSClient) est créée et connectée à la socket du client demandeur et dont l'adresse est stockée dans adClient

Accepter une demande de connexion d'un client

```
int accept(int descr,                /* descripteur de la socket recevant des
                                        demandes de connexion*/
    struct sockaddr *adrClient,      // pointeur vers l'adresse de la socket du client
    socklen_t lgAdr)                 // pointeur vers longueur de l'adresse
```

Exemple

```
... // code incluant la création d'une socket (descripteur dS)
int res = listen(dS, 10);
struct sockaddr_in adClient; socklen_t lgA = sizeof(struct sockaddr_in);
int dSClient = accept(dS, (struct sockaddr *) &adClient, &lgA);
```

Extrait une demande de connexion de la socket dS et la traite. En cas de succès, une nouvelle socket (descripteur dSClient) est créée et connectée à la socket du client demandeur et dont l'adresse est stockée dans adClient

Valeur de retour

Si demande acceptée, le descripteur (> 0) de la socket créée, -1 sinon (avec errno positionné).

Réception d'un message

```
ssize_t recv (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet où
                                       sera stocké le message reçu */
    size_t lg,                      // le nombre max d'octets attendus
    int flags)                      // options de réception, 0 par défaut
```

Réception d'un message

```
ssize_t recv (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet où
                                       sera stocké le message reçu */
    size_t lg,                      // le nombre max d'octets attendus
    int flags)                      // options de réception, 0 par défaut
```

Valeur de retour

Si l'appel réussi, le nombre d'octets effectivement extraits (> 0 et $\leq lg$), 0 si la socket a été fermée, -1 sinon (avec errno positionné).

Réception d'un message

```
ssize_t recv (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet où
                                       sera stocké le message reçu */
    size_t lg,                      // le nombre max d'octets attendus
    int flags)                      // options de réception, 0 par défaut
```

Valeur de retour

Si l'appel réussi, le nombre d'octets effectivement extraits (> 0 et $\leq lg$), 0 si la socket a été fermée, -1 sinon (avec errno positionné).

Note

Cette fonction est bloquante. Dans quel cas ?

Réception d'un message - exemple

Prérequis

- Une socket correctement créée et connectée à une socket distante.
- *Espace mémoire alloué (statiquement ou dynamiquement) pour stocker le message reçu.*

Réception d'un message - exemple

Prérequis

- Une socket correctement créée et connectée à une socket distante.
- *Espace mémoire alloué (statiquement ou dynamiquement) pour stocker le message reçu.*

Que fait le code suivant ?

```
... // code incluant la création d'une socket dS et sa connexion
int message [500] ;
ssize_t res = recv(dS, message, sizeof(message), 0) ;
```

Réception d'un message - exemple

Prérequis

- Une socket correctement créée et connectée à une socket distante.
- *Espace mémoire alloué (statiquement ou dynamiquement) pour stocker le message reçu.*

Que fait le code suivant ?

```
... // code incluant la création d'une socket dS et sa connexion
int message [500] ;
ssize_t res = recv(dS, message, sizeof(message), 0) ;
```

Questions :

- Que signifie le commentaire "création d'une socket dS et sa connexion" coté client ?
- et coté serveur ?

Envoi d'un message

```
ssize_t send (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet
                                       du message à envoyer */
    size_t lg,                      // le nombre d'octets du message
    int flags)                      // options d'envoi, 0 par défaut
```

Envoi d'un message

```
ssize_t send (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet
                                       du message à envoyer */
    size_t lg,                      // le nombre d'octets du message
    int flags)                      // options d'envoi, 0 par défaut
```

Valeur de retour

Si l'appel réussi, le nombre d'octets (> 0 et $\leq lg$) effectivement déposés dans le buffer associé à la socket, 0 si la socket a été fermée, -1 sinon (avec errno positionné).

Envoi d'un message

```
ssize_t send (int descripteur,      // descripteur de socket
    const void *msg,                /* pointeur vers le premier octet
                                       du message à envoyer */
    size_t lg,                      // le nombre d'octets du message
    int flags)                      // options d'envoi, 0 par défaut
```

Valeur de retour

Si l'appel réussi, le nombre d'octets (> 0 et $\leq lg$) effectivement déposés dans le buffer associé à la socket, 0 si la socket a été fermée, -1 sinon (avec errno positionné).

Note

Cette fonction est bloquante. Dans quel cas ?

Envoi d'un message - exemple

Prérequis

- Une socket correctement créée et connectée ou en attente de connexion.
- Le message à envoyer correctement construit.

Envoi d'un message - exemple

Prérequis

- Une socket correctement créée et connectée ou en attente de connexion.
- Le message à envoyer correctement construit.

Que fait le code suivant ?

```
... // code incluant la création d'une socket dS et sa connexion
int message [500] ;
saisiValsTab(message, 500) ; // saisie des éléments du tableau
ssize_t res = send(dS, message, sizeof(message), 0) ;
```

Envoi d'un message - exemple

Prérequis

- Une socket correctement créée et connectée ou en attente de connexion.
- Le message à envoyer correctement construit.

Que fait le code suivant ?

```
... // code incluant la création d'une socket dS et sa connexion
int message [500] ;
saisiValsTab(message, 500) ; // saisie des éléments du tableau
ssize_t res = send(dS, message, sizeof(message), 0) ;
```

Question

Si la valeur de retour *res* est 1024, que s'est-il produit ? Que doit on faire pour que les 500 entiers soient envoyés ?

On reprend tout : exemple

Client (extrait)

```
dS= socket(PF_INET, SOCK_STREAM, 0) ;
struct sockaddr_in aS ;
aS.sin_family = AF_INET ;
inet_pton(AF_INET,argv[1],&(aS.sin_addr)) ;
aS.sin_port = htons(atoi(argv[2])) ;
socklen_t lgA = sizeof(struct sockaddr_in) ;
connect(dS, (struct sockaddr *) &aS, lgA) ;

char * m = "Bonjour" ;
send(dS, m, 8, 0) ;
int r ;
recv(dS, &r, sizeof(int), 0) ;
printf("reponse : %d", r) ;
close (dS) ;
```

Serveur (extrait)

```
dS= socket(PF_INET, SOCK_STREAM, 0) ;
struct sockaddr_in ad ;
ad.sin_family = AF_INET ;
ad.sin_addr.s_addr = INADDR_ANY ;
ad.sin_port = htons(atoi(argv[1])) ;
bind(dS, (struct sockaddr*)&ad, sizeof(ad)) ;
listen(dS, 7) ;
struct sockaddr_in aC ;
socklen_t lg = sizeof(struct sockaddr_in) ;
dSC= accept(dS, (struct sockaddr*) &aC,&lg) ;
char msg [20] ;
recv(dSC, msg, sizeof(msg), 0) ;
printf("recu : %s", msg) ;
int r = 10 ;
send(dSC, &r, sizeof(int), 0) ;
close (dSC) ; close (dS) ;
```

+ gestion des retours de fonctions !

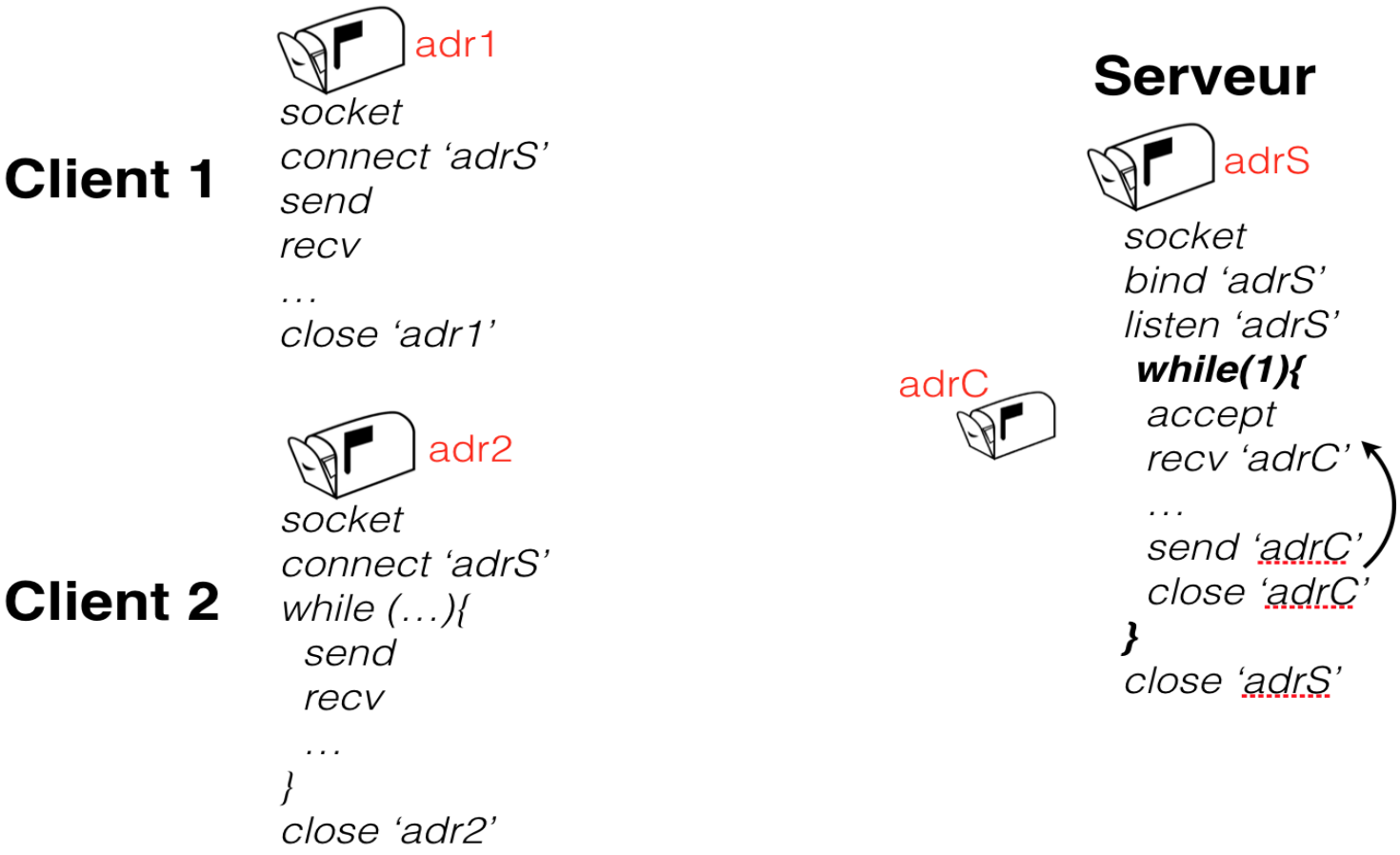
Pour finir

- Un message en TCP est vu comme *un flux d'octets et peut être reçu et envoyé en une ou plusieurs fois.*
  - *si un appel à send(...) ou recv(...) retourne un nombre d'octets inférieur au nombre attendu, il ne s'agit pas d'une erreur !* Ce cas est à prendre en compte à chaque utilisation de ces fonctions.
- Il n'est pas nécessaire d'être en réception pour recevoir un message.
- En TCP, il est aussi possible d'utiliser les fonctions *read(...)* et *write(...)* pour recevoir et envoyer des messages, ainsi que *sendto(...)*, *recvfrom(...)*.
- Les appels des fonctions : *connect(...)*, *accept(...)*, *send(...)*, *recv(...)*, *read(...)*, *write(...)*, *sendto(...)* et *recvfrom(...)* *peuvent être bloquants* .

Introduction

- Les sections précédentes ont illustré l'utilisation des protocoles TCP dans le cas d'un seul client.
- Question actuelle : comment prendre en compte plusieurs clients ?
- Deux types de serveurs :
  - itératif *: traite un client à la fois et l'un(e) après l'autre. Il y a aussi la possibilité de traiter plusieurs clients en même temps mais une requête après l'autre via le multiplexage des entrées/sorties*
  - concurrent *: traite plusieurs clients (TCP) requêtes en parallèle (simultanément).*

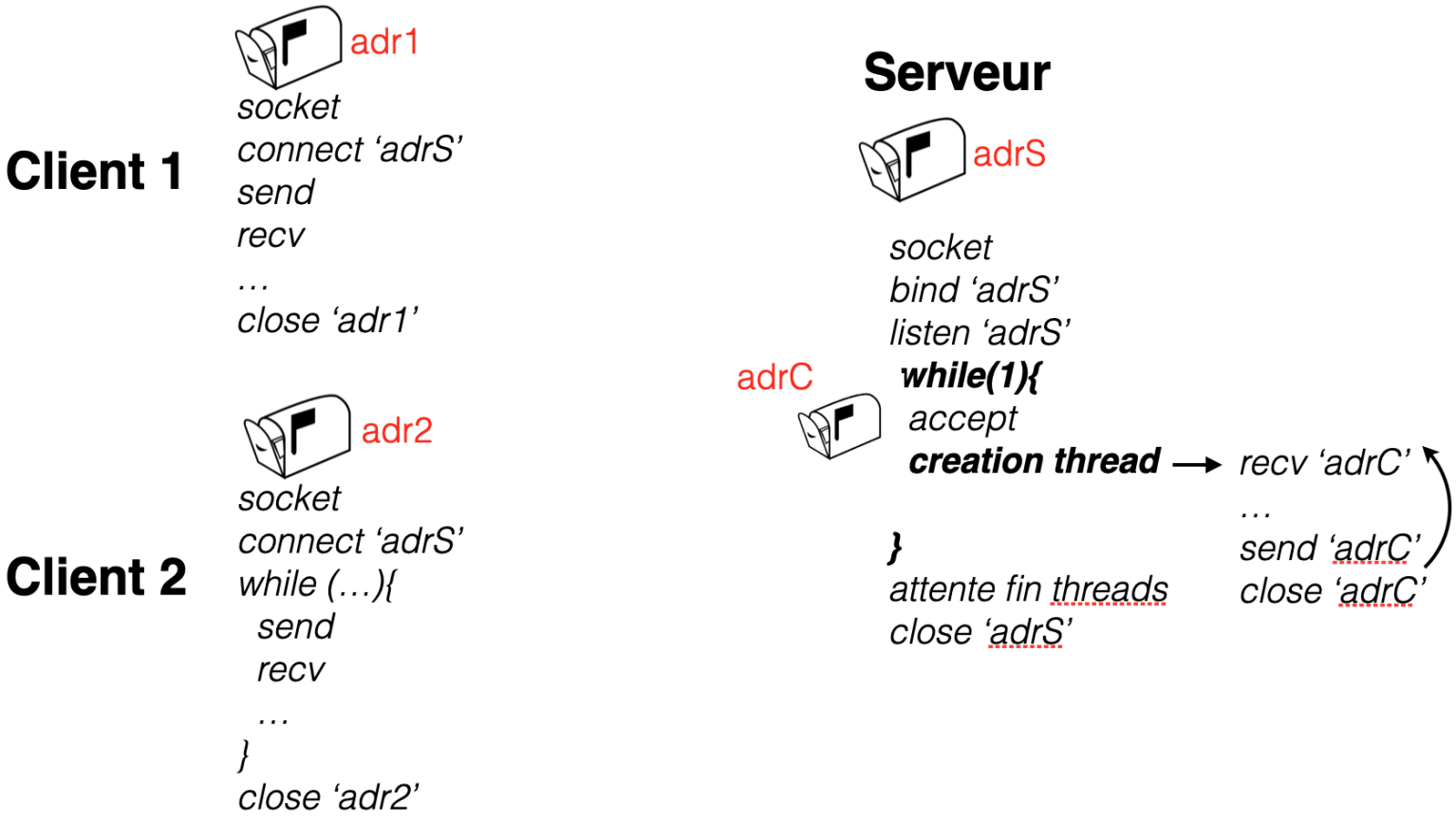
Serveur itératif - TCP



Note

Fermer la socket dédiée à un client à la fin du traitement de ce dernier.

Serveur concurrent multi-threads - TCP



Note

C'est une solution parmi d'autres. Quelles sont vos idées ?