



Programmation Client/Serveur

Les Sockets

Ahmed ZELLOU

Ahmed.zellou@um5.ac.ma

Sup-MTI, 2023-2024.

Quelques Règles



Pas de GSM



Pas de Bavards



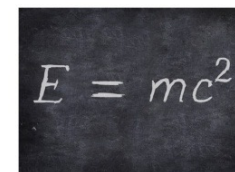
Poser des Questions



Partager votre expérience



Tout est relatif



Plan

- Notions générales
- Présentation sockets
- Caractéristiques d'une socket
- Primitives d'une socket
- Sockets en mode connecté
- Envoyer et recevoir les données
- Sockets en mode non connecté
- Processus père et Processus fils.

Notions générales

- Programmation réseau
 - Permettre à des **processus** de communiquer (échanger des données) avec d'autres processus se trouvant sur des ordinateurs locaux ou distants (connectés par un réseau).
 - Plusieurs outils : mémoire partagée, **sockets**.
- La mise en œuvre de la communication entre processus nécessite des **protocoles de communication**.

Notions générales

- Protocole
 - Un protocole est un **ensemble d'accords** sur:
 - **début** d'un message
 - la **représentation** des bits
 - détection de la **fin** d'un message
 - **acheminement** des message
 - **représentation** des nombres, des caractères
 - etc ...
- Il faut un accord à plusieurs niveaux, depuis les détails de bas niveau de la transmission des bits jusqu' à ceux de plus haut niveau de la représentation des données.

Notions générales

- Modèle client/serveur
 - La majorité des applications réseau modernes se fondent sur le modèle **client/serveur**.
 - Serveur :
 - Un programme qui s'exécute sur une machine et qui est capable d'offrir un **service**.
 - Le serveur **reçoit** une requête à travers le réseau, effectue le traitement nécessaire et **retourne** le résultat de la requête.
 - Client : un programme qui **envoie** une requête au serveur et **reçoit** une réponse.

Notions générales

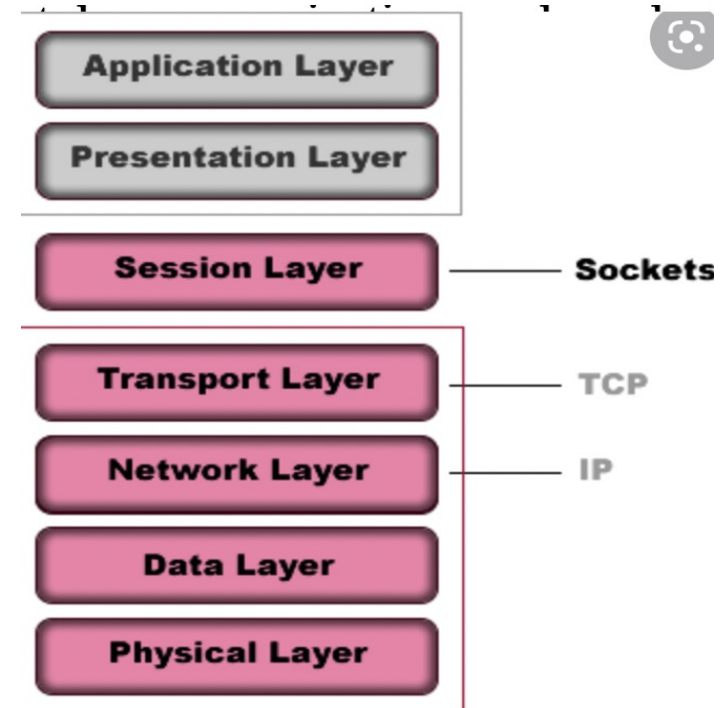
■ Ports

- Plusieurs applications réseau peuvent tourner sur la même machine.
- Le numéro de **port** permet d'identifier l'application avec laquelle on souhaite communiquer.
- Un port est un entier sur 16 bits:
 - 65536 possibilités.
 - Les ports < 1024 sont réservés (standards).
- Les numéros de ports standards sont définis dans le fichier /etc/services, et accessibles via la commande `getservbyname()`.

Sockets

■ Présentation

- Une socket (**prise**) est un point de communication par lequel un processus peut émettre ou recevoir des informations.
- La programmation par les sockets se base sur un modèle client/serveur.
- Les sockets se situent juste au-dessus de la couche transport du modèle OSI (utilisant les protocoles TCP ou UDP).

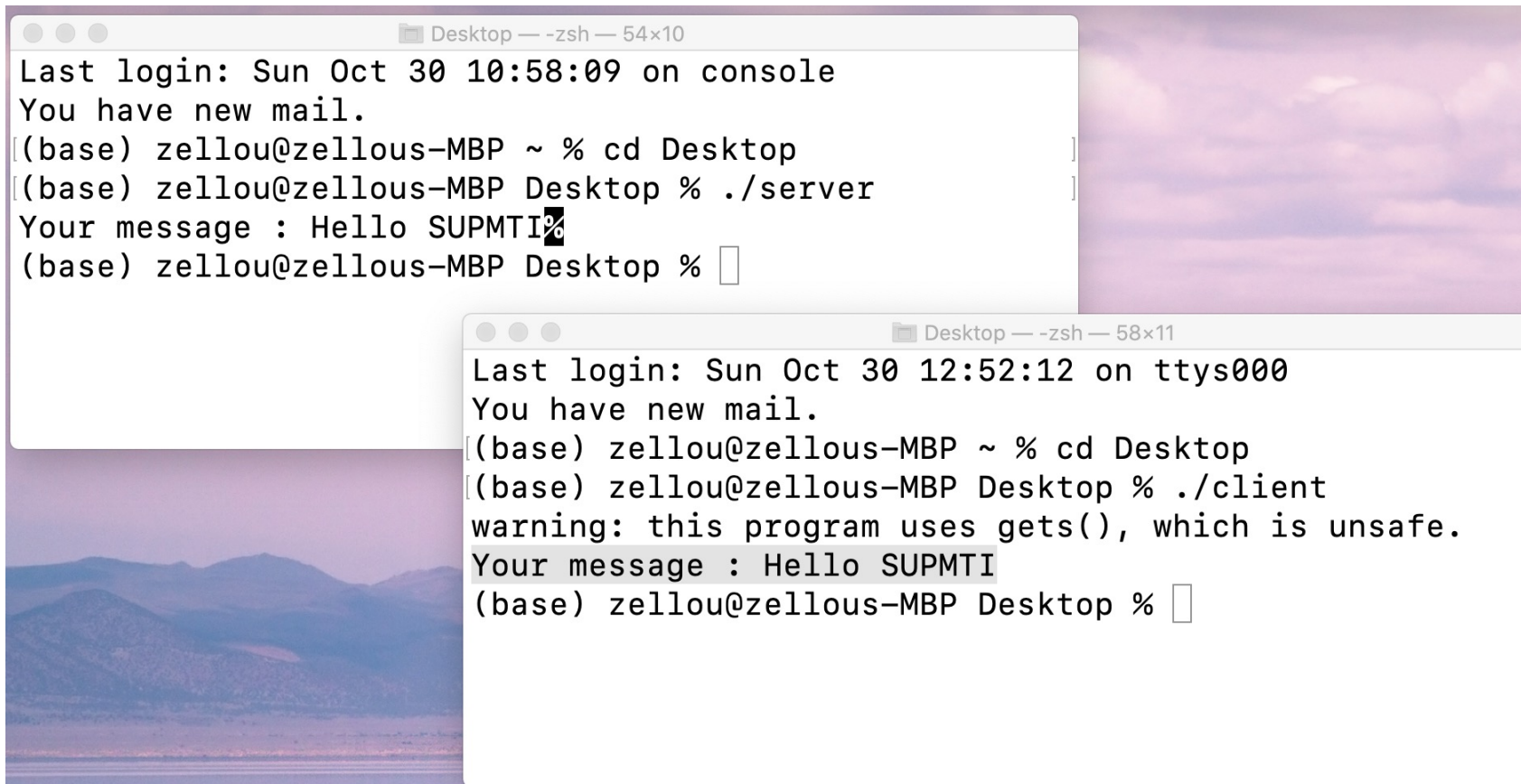


Sockets

- Principe de base

1. Chaque **processus** crée une socket,
2. Chaque socket sera associée à un **port** de la machine sur laquelle se déroule le processus qui l'a créée,
3. Les deux sockets seront explicitement **connectées** si on utilise un protocole en mode connecté,
4. Chaque processus **lit** et/ou **écrit** dans sa socket,
5. Les données sont **envoyées** d'une socket à une autre à travers le réseau,
6. Une fois terminé, chaque processus **ferme** sa socket.

Test

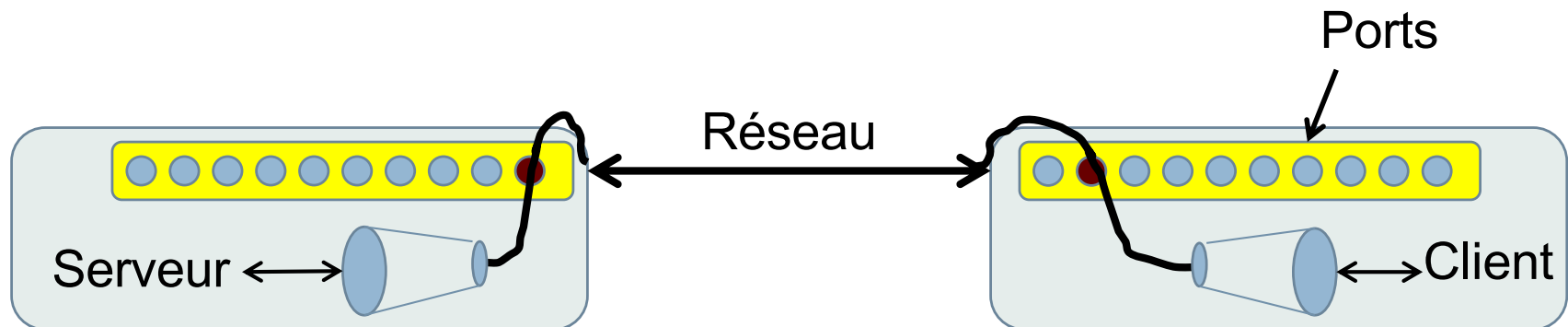


```
Desktop — -zsh — 54x10
Last login: Sun Oct 30 10:58:09 on console
You have new mail.
(base) zellou@zellous-MBP ~ % cd Desktop
(base) zellou@zellous-MBP Desktop % ./server
Your message : Hello SUPMTI%
(base) zellou@zellous-MBP Desktop %

Desktop — -zsh — 58x11
Last login: Sun Oct 30 12:52:12 on ttys000
You have new mail.
(base) zellou@zellous-MBP ~ % cd Desktop
(base) zellou@zellous-MBP Desktop % ./client
warning: this program uses gets(), which is unsafe.
Your message : Hello SUPMTI
(base) zellou@zellous-MBP Desktop %
```

Sockets

- Analogie
 - Une socket peut être vue comme une **boîte aux lettres** ou un **poste téléphonique**.



Caractéristiques d'une socket

- **Domaine** 1 / 3

- Il spécifie le format des adresses possibles et par conséquent l'ensemble des sockets qui peuvent être atteintes.
 - **AF_UNIX** : les sockets sont locaux au système. Elles permettent la communication entre processus appartenant au même système (machine).
 - **AF_INET** : Permet la communication entre processus appartenant à différentes machines.

Caractéristiques d'une socket

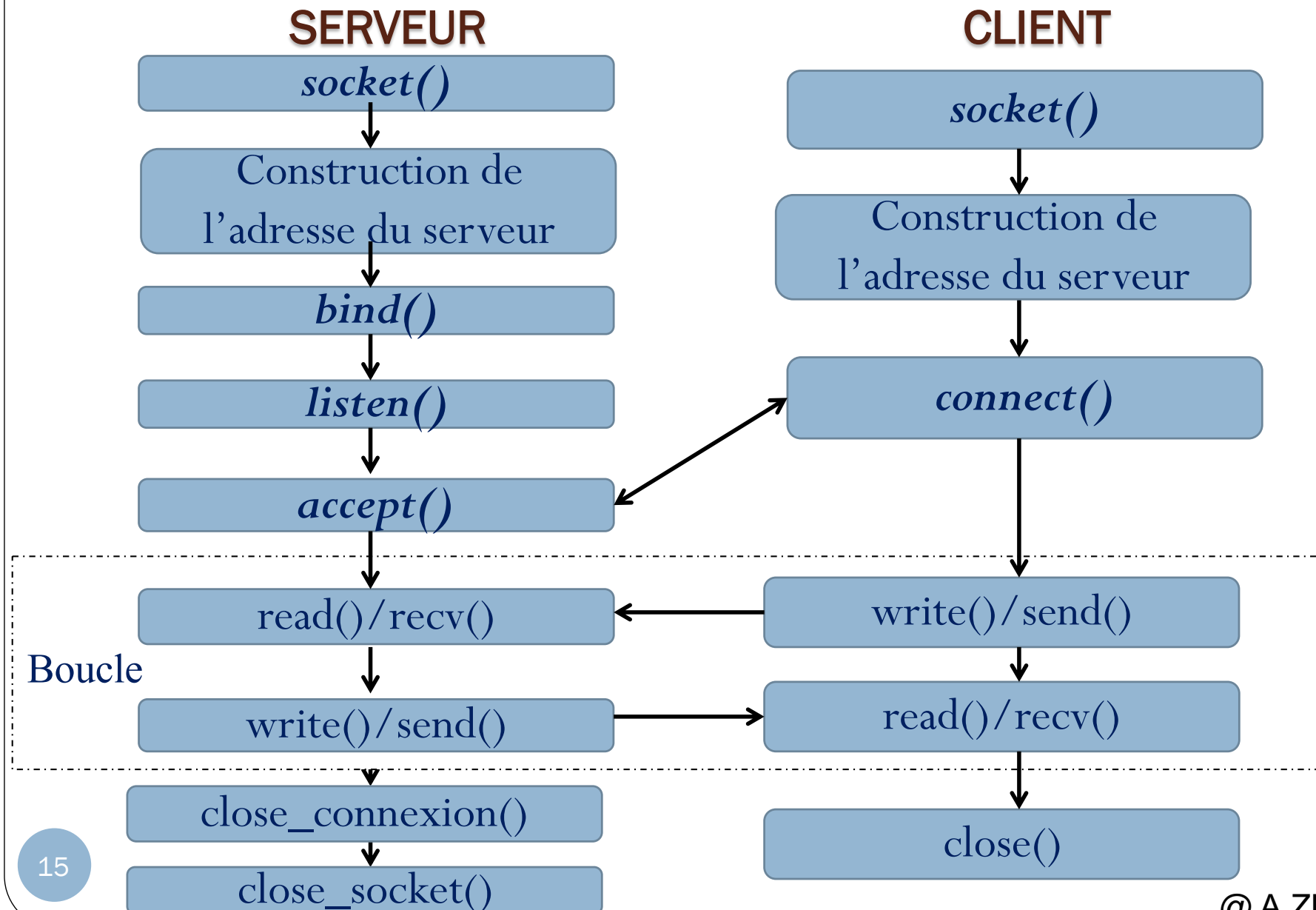
- **Type** 2 / 3
 - Il définit la nature des communications supportées par la socket, les deux les plus répandus :
 - **SOCK_STREAM** : communication en mode connecté par l'envoi de flots d'informations (Téléphone).
 - **SOCK_DGRAM** : communication en mode non connecté par l'envoi de datagrammes (boîte aux lettres).

Caractéristiques d'une socket

- **Protocole** 3/3

- Il spécifie le protocole de Transport utilisé par la socket.
- Deux protocoles peuvent être utilisés:
 - **TCP** (Transmission Control Protocol) : Permet d'établir une communication fiable entre deux applications avec gestion d'erreur et contrôle de flux.
 - **UDP** (User Datagram Protocol): permet d'envoyer des messages en mode non connecté.

Sockets en mode connecté



Primitives d'une socket

- Création d'une socket

```
#include <sys/socket.h>
int socket (int domaine,
            int type,
            int protocole)
```

- **Domaine**: AF_INET, AF_UNIX
 - **Type**: SOCK_DGRAM, SOCK_STREAM
 - **Protocole**: TCP, UDP ou 0 (laisse le système choisir lui-même le bon protocole)
- La valeur de retour de la primitive socket est un descripteur qui permet d'accéder à la socket créée.

Primitives d'une socket

- Création d'une socket

```
int sd;  
sd=socket(AF_INET,  
          SOCK_STREAM,  
          0);  
if (sd<0)  
    printf("erreur de création\n" );
```

- Retourne -1 en cas d'erreur

Primitives d'une socket

- Adresse d'une socket
 - La structure de l'adresse est la suivante :

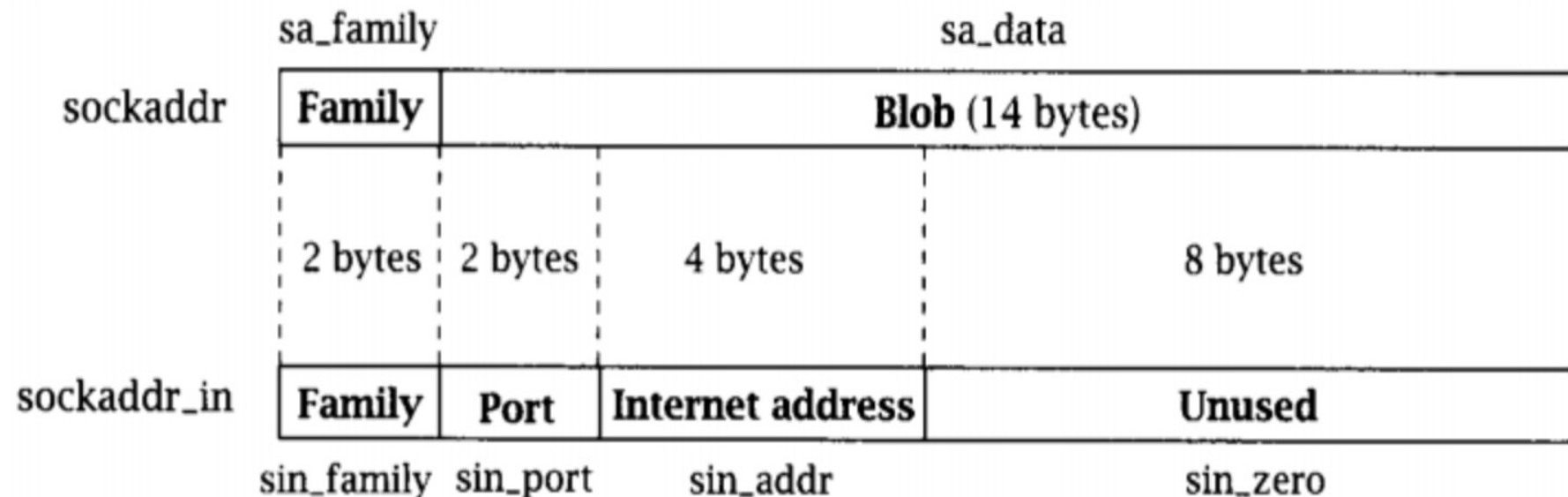
```
struct sockaddr_in {  
    short sin_family;           /* AF_INET */  
    u_short sin_port;           /* N° port */  
    struct in_addr sin_addr;    /* @IP */  
    char sin_zero[8];           /* non utilisé */  
}
```

```
struct in_addr {  
    u_long s_addr;              /* Pour l'adresse IP */  
}
```

- Définie dans la bibliothèque `<netdb.h>`

Primitives d'une socket

- Adresse d'une socket
 - La structure de l'adresse est la suivante :

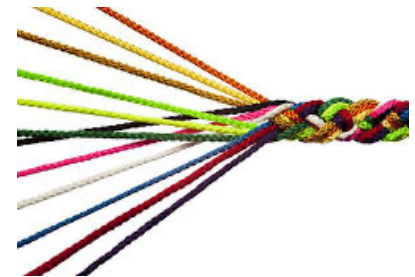


- INADDR_ANY
 - Une adresse IP libre, utilisée quand on ne connaît pas l'adresse de la machine.
 - Elle permet de recevoir mes paquets adressés à toutes les interfaces.

Primitives d'une socket

- Attachement d'une socket

```
#include <sys/socket.h>
int bind(int sd,
         struct sockaddr_in *p_adresse,
         int lg_adresse)
```



- **sd** : Descripteur de la socket
 - ***p_adresse**: pointeur sur l'adresse
 - **lg_adresse**: Longueur de l'adresse
- La primitive **bind** renvoie 0 en cas de réussite et -1 sinon.

Primitives d'une socket

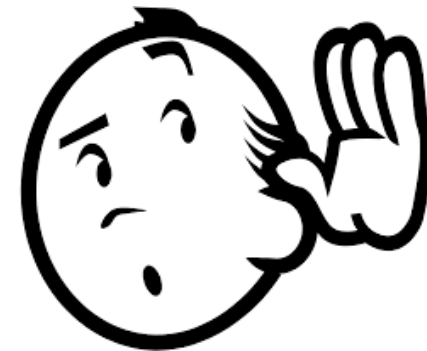
- Attachement d'une socket

```
int sd;  
struct sockaddr_in adresse;  
  
sd=socket(AF_INET, SOCK_STREAM, 0);  
if (sd<0)  
{printf ("erreur de création\n"); exit(0);}  
  
adresse.sin_family = AF_INET;  
adresse.sin_port = 1230;  
adresse.sin_addr.s_addr = INADDR_ANY;  
bind (sd, (struct sockaddr *) &adresse, sizeof(adresse));
```

Primitives d'une socket

- Ouverture du service

```
#include <sys/socket.h>  
int listen (int sd,  
           int nb)
```



- **sd**: Descripteur de la socket
 - **nb**: Nombre maximal de demandes de connexion pendantes
-
- La primitive **listen** renvoie 0 en cas de succès.
 - Elle permet de signaler au système qu'il est prêt à accepter des demandes de connexion.

Primitives d'une socket

■ Demande de Connexion

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

int connect (int sd,
             struct sockaddr_in *p_adresse,
             int lg_adresse)
```

- **sd**: Descripteur de la socket
 - ***p_adresse**: adresse de la socket distante
 - **lg_adresse**: Longueur de l'adresse
-
- La primitive **connect** renvoie 0 en cas de succès.
 - Elle est utilisée pour demander la connexion au serveur.



Primitives d'une socket

- Echec de Connexion

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

int connect (int sd,
             struct sockaddr_in *p_adresse,
             int lg_adresse)
```

- La cause d'un échec de demande de connexion peut être :
 - Les paramètres reçus par **connect** sont incorrects
 - **p_adresse** est déjà utilisée dans une connexion
 - La file des connexions de la socket distante est pleine

Primitives d'une socket

- Acceptation d'une requête

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

int accept(int sd,
           struct sockaddr_in *p_adresse,
           int lg_adresse)
```

- **sd**: Descripteur de la socket
- ***p_adresse**: adresse de la socket distante (adresse du client)
- **lg_adresse**: Longueur de l'adresse

Primitives d'une socket

- Acceptation d'une requête

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

int accept(int sd,
           struct sockaddr_in *p_adresse,
           int lg_adresse)
```

- La primitive **accept** permet d'extraire une connexion dans la file d'attente associée de la socket pour laquelle un appel à **listen** a été réalisé.
- Elle retourne un descripteur de connexion appelé socket de service par laquelle la communication va se dérouler avec le client.

Primitives d'une socket

- Acceptation d'une requête

```
#include <sys/socket.h>  
ns=accept(sd,0,0);
```

- Nous pouvons récupérer l'adresse du client part la suite via **getpeername()**

Envoie de données

- Envoie de données

```
#include <unistd.h>

write(int ns,
      char * message,
      int sizeof("message"));
```

- **sd**: Descripteur de la connexion ou de la socket
- **message** : le message à envoyer
- **sizeof("message")** : taille du message
- La primitive **write** permet d'envoyer un message.
- Le client écrit sur la socket et le serveur écrit sur la connexion.
- Retourne le nombre d'octets envoyés au cas de succès et -1 au cas d'erreur.

Réception de données

- Envoie de données

```
#include <unistd.h>

int read(int sd,
        char* message,
        int size_t message);
```



- **sd**: Descripteur de la socket
- **message** : le message à lire
- **sizeof("message")** : taille du message
- La primitive **read** permet de lire un message.
- Le client lit sur la socket et le serveur lit sur la connexion.
- Retourne le nombre d'octets lu au cas de succès et -1 au cas d'erreur.

Envoie de données

- Envoie de données

```
send(int sd,  
      char* message,  
      int len,  
      int flags);
```

- **sd**: Descripteur de la connexion
- **message**: le message à envoyer
- **len**: taille du message
- **flags**: initialisé à 0
- La primitive **send** permet d'envoyer un message dans le descripteur de la socket/connexion.
- Retourne 0 au cas de succès et -1 au cas d'erreur.

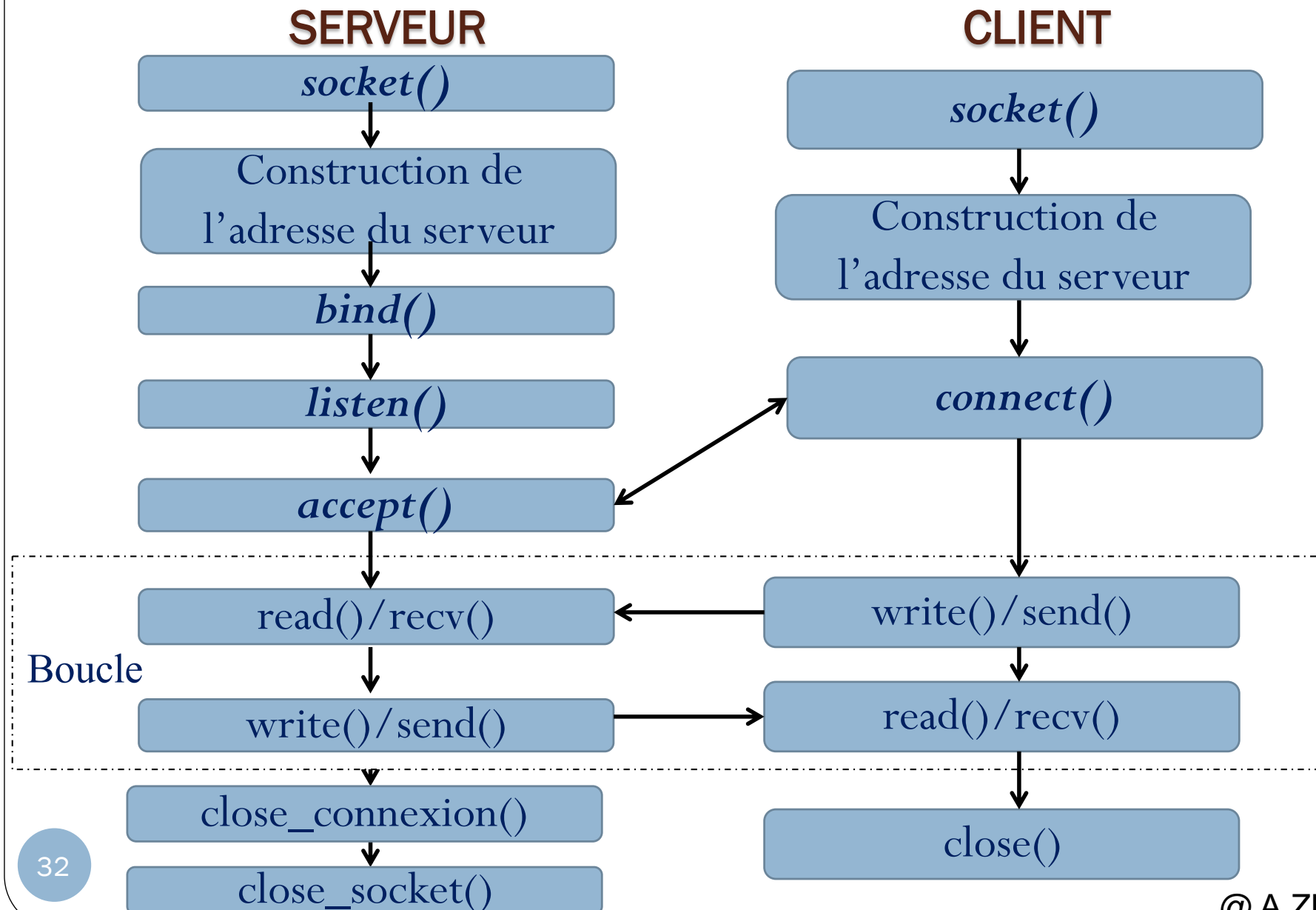
Réception de données

- Envoie de données

```
recv(int sd,  
      char* message,  
      int len,  
      int flags);
```

- **sd**: Descripteur de la connexion
- **message**: le message reçu
- **len**: taille du message
- **flags**: initialisé à 0
- La primitive **recv** permet de recevoir un message dans le descripteur de la socket/connexion.
- Retourne 0 au cas de succès et -1 au cas d'erreur.

Sockets en mode connecté



Atelier

- Envoie de données
 - Rédigez un programme en mode connecté qui envoie un message du serveur vers le client.

Etapes du Serveur

1. *appels des bibliothèques*
2. *la fonction main*
3. *déclaration des variables*
4. *socket*
5. *préparation de l'adresse serv*
6. *bind*
7. *listen*
8. *accept*
9. *read du message*
10. *affichage du message*
11. *close de la connexion*
12. *close de la socket*

Etapes du Client

1. *appels des bibliothèques*
2. *la fonction main*
3. *déclaration des variables*
4. *socket*
5. *préparation de l'adresse serv*
6. *connect*
7. *write du message*
8. *affichage du message*
9. *close()*

Atelier

- Envoie de données
 - Rajouter la confirmation de la réception.

Etapes du Serveur

1. *appels des bibliothèques*
2. *la fonction main*
3. *déclaration des variables*
4. *socket*
5. *préparation de l'adresse serv*
6. *bind*
7. *listen*
8. *accept*
9. *read du message*
10. *affichage du message*
11. *write du message*
12. *close de la connexion*
13. *close de la socket*

Etapes du Client

1. *appels des bibliothèques*
2. *la fonction main*
3. *déclaration des variables*
4. *socket*
5. *préparation de l'adresse serv*
6. *connect*
7. *write du message*
8. *read du message*
9. *affichage du message*
10. *close()*

Atelier

- Envoie de données
 - Rédigez un programme en mode connecté qui permet à un serveur de traiter plusieurs clients en série.
 - Modifiez le programme pour permettre au serveur de traiter plusieurs clients en parallèle.

fork()

- Crée un nouvel processus, appelé fils, qui s'exécute au même temps que le processus appelant, appelé père.

```
#include<unistd.h>
pid_t fork ( );
```

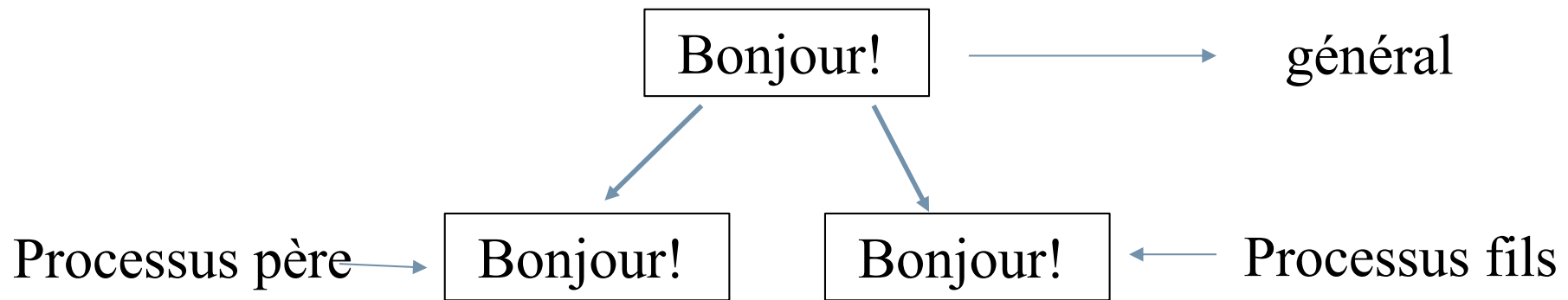
- Le processus fils commence l'exécution à partir de l'appel.
- Les deux processus ont accès aux mêmes ressources : variables, fichiers, ...
- fork() retourne une valeur entière :
 - **Négative** : si la création d'un processus enfant a échouée.
 - **0** : un pointeur sur le procesus fils.
 - **Valeur positive** : un pointeur sur le procesus père.

fork()

■ Exemple :

```
printf("Bonjour!\n");  
fork();  
printf("Bonjour!\n");
```

```
(base) zellou@zellous-MBP Desktop % gcc -o fork fork.c  
(base) zellou@zellous-MBP Desktop % ./fork  
Bonjour!  
Bonjour!  
Bonjour!  
(base) zellou@zellous-MBP Desktop %
```



Atelier

- Envoie de données
 - Rédigez un programme en mode connecté qui permet à un serveur de traiter plusieurs clients en parallèles.

Etapas du Serveur

1. *socket*
2. *bind*
3. *listen*
4. *accept*
5. *fork*
6. *recv*
7. *send*
8. *close*

Boucle

Etapas du Client

1. *socket*
2. *connect*
3. *send*
4. *recv*
5. *close*

Atelier FTP

- Envoie de données
 - Écrivez un programme qui permet à un client d'envoyer un fichier vers le serveur selon l'algorithme suivant :

Algorithme du Client

```
socket  
connect  
ouvrir le fichier en mode lecture  
envoyer un message FILE qui indique l'envoi d'un fichier  
envoyer un message contenant le nom du fichier  
loop  
    lire l'ensemble des lignes  
    envoyer chaque ligne lue au serveur  
fin loop  
fermer le fichier  
envoyer un message EOF de fin d'envoi  
fermer la connexion
```

Atelier FTP

- Envoie de données
 - Écrivez un programme qui permet à un client d'envoyer un fichier vers le serveur selon l'algorithme suivant :

Algorithme du Serveur

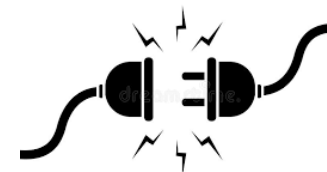
```
socket, bind, listen, accept  
si reception du message FILE qui indique l'envoie d'un fichier  
réception du message contient la nom du fichier à créer  
ouvrir le fichier en mode écriture  
loop  
pour chaque message reçu  
écrivez le message dans le fichier si message différent de EOF  
fin loop si message est EOF  
fermer le fichier  
fin si  
fermer la connexion
```


Node non connecté

- **Moins fiable** que le mode connecté.

```
socket(AF_INET, SOCK_DGRAM, 0);
```

- Ne nécessite pas de **création** ni de **terminaison** de connexion.
- Chaque message est traité comme une **entité appart**.
- Chaque message porte **l'adresse de destination** pour identifier le récepteur.
- Les messages envoyés ne suivent **pas le même chemin**.
- **L'ordre** de réception n'est pas toujours le même que celui d'envoi.
- Adapté pour une transmission par **rafale**.

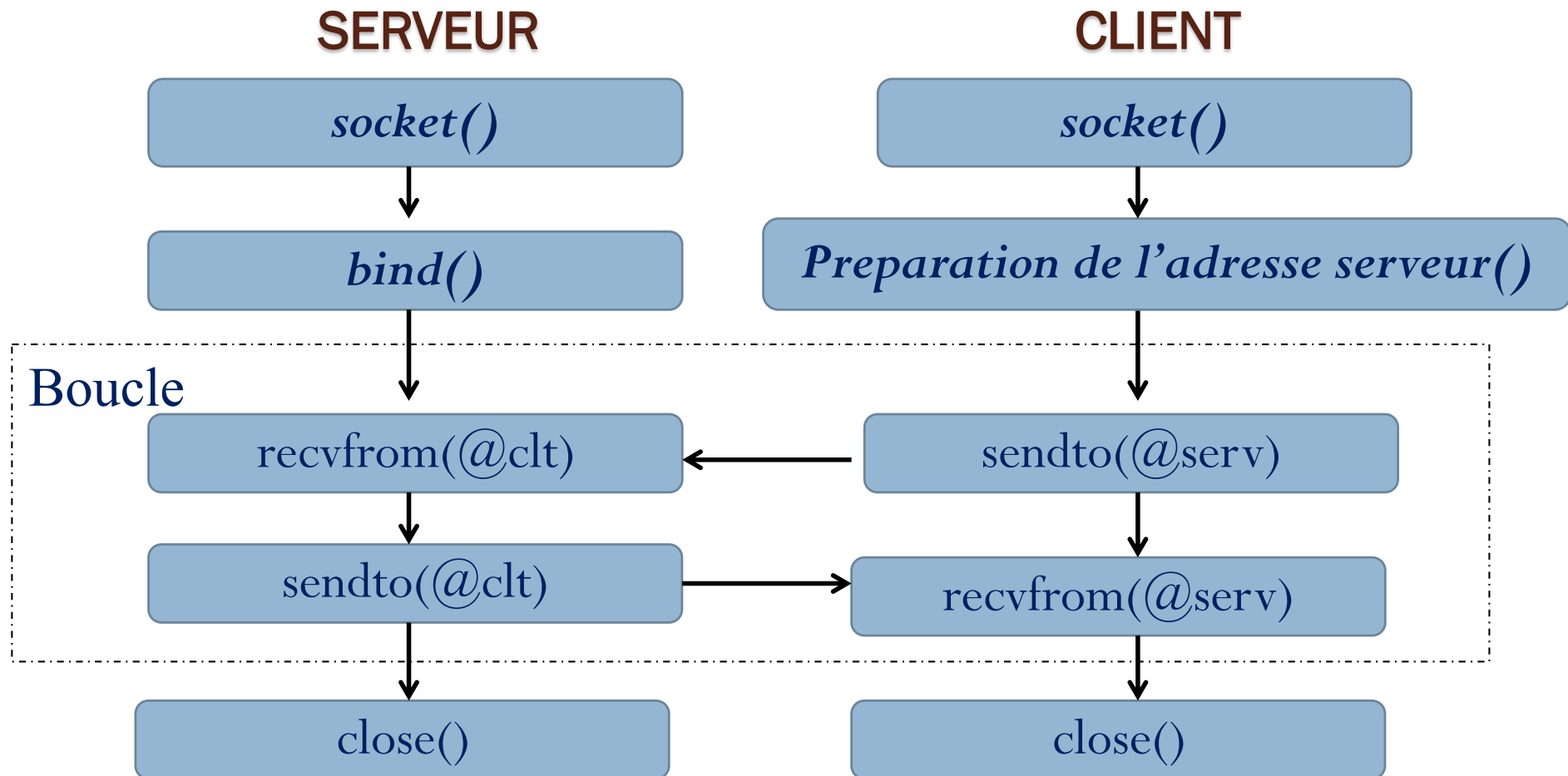


Connecté versus non Connecté



| Mode | Connecté | Non connecté |
|---------------------|-----------------------------------|--------------------------|
| Connexion préalable | Requise | Non requise |
| Fiabilité | Oui | non garantie |
| Ressources | Besoin d'être attribuées | Non attribuées |
| Retard | Communication longue | Communication rapide |
| Routage | Identique | Non Identique |
| Signalisation | Utilisé pour établir la connexion | Pas de signalisation |
| Utilisation | Communication régulière | Communication par rafale |
| Données perdues | Retransmission Faisable | Pas de retransmission |
| Mode de transfert | Commutation de circuit | Commutation de paquets |
| Lancement | Serveur avant | Pas d'ordre |
| Congestion | Peu probable | Très probable |

Sockets en mode non connecté



Envoie de données

- **Envoie** de données (mode non connecté)

```
sendto(int sd,           // descripteur de la connexion/socket
char* message, // le message à envoyer
int len,        // taille du message
int flags;      // initialisé à 0
const struct sockaddr *dest_addr, // @ destinataire
socklen_t addrlen // taille de l'adresse destinataire
);
```

- La primitive **sendto** permet d'envoyer un message dans le descripteur de la socket/connexion.
- Retourne le nombre d'octets envoyés au cas de succès et -1 au cas d'erreur.

Réception de données

- **Réception** de données (mode non connecté)

```
recvfrom (int sd,          // descripteur de la connexion/socket
          char* message, // le message reçu
          int len,         // taille du message
          int flags;       // initialisé à 0
          const struct sockaddr *dest_addr, // @ de la source
          socklen_t addrlen // taille de l'adresse de la source
);
```

- La primitive **recvfrom** permet de recevoir un message dans le descripteur de la socket/connexion.
- Retourne le nombre d'octets reçu, 0 si le destinataire est arrêté et -1 au cas d'erreur.

Atelier

- Envoie de données
 - Rédigez un programme en mode non connecté qui permet à deux programmes client et serveur de communiquer.

SERVEUR

1. *socket*
2. *bind*

Boucle

3. *recvfrom()*
4. *sendto()*

5. *close*

CLIENT

1. *socket*
2. *prep @serv*

3. *sendto()*
4. *recvfrom()*

5. *close*

Gestion d'adresse

- **getpid** : process courant

```
#include <unistd.h>  
pid_t getpid(void);
```

- Retourne l'identifiant id du processus appelant.

- **getppid** : Processus parent

```
#include <unistd.h>  
pid_t getppid(void);
```

- Retourne le PID (Process Identifier) du processus parent.

Gestion d'adresse

- `inet_addr` : Conversion d'adresse

```
#include <arpa/inet.h>
in_addr_t inet_addr(const char *cp);
```

- Convertit la chaîne `cp` contenant une adresse IPv4 vers un entier, à utiliser dans l'adresse internet.
 - `in_addr_t` : une structure que contient au moins `s_addr` de type `u_long`.

```
serveraddr.sin_addr.s_addr=inet_addr("192.168.0.160");
```

- Autres fonctions :
 - `inet_network`, `inet_makeaddr`, `inet_netof`, ...

Gestion d'adresse

- **inet_aton** : Conversion d'adresse

```
#include <arpa/inet.h>
int inet_aton(const char *source, struct in_addr *cible);
```

- Convertit l'adresse internet de IPV4 vers un format binaire
- Retourne 0 si l'adresse n'est pas valide
 - **source** : l'adresse au format String
 - **cible** : structure de type in_addr

```
struct in_addr addr;
if (inet_aton("142.250.200.100", &addr) == 0) {
    fprintf(stderr, "Adresse non valide\n"); }
else printf("Adresse est bonne");
printf("Adresse au format binaire : %s\n", inet_ntoa(addr));
```

- Autres fonctions :
 - inet_aton, inet_addr, inet_ntoa, inet_makeaddr,...

Les données d'une adresse

- `gethostbyaddr` : résolution DNS

```
#include <netdb.h>
struct hostent *gethostbyaddr(const void *addr,
socklen_t len, int type);
```

- Retourne une structure de type `hostent` que contient : l'adresse IP, le nom officiel, les aliases, le type d'adresse, la taille des adresses et la liste des adresses.
- Utilisable avec `AF_INET` et `AF_INET6`.

```
struct hostent* host;
host = gethostbyaddr( (const void*)&servaddr.sin_addr, sizeof(struct
in_addr), AF_INET );
printf( "Connection de %s: \n", host->h_name );
```

Les données d'une adresse

■ gethostbyaddr : Exemple

```
struct hostent *he;
struct in_addr **addr_list;
struct in_addr address;
printf("Le process numéro : %d\n", getpid());
printf("Le process parent numéro : %d\n", getppid());

const char* ip_address = "142.250.184.164";
struct hostent* host;
inet_pton(AF_INET, ip_address, &address);
host = gethostbyaddr(&address, sizeof(struct in_addr), AF_INET);

printf( "Connection de %s: \n", host->h_name );
```

Les données d'une adresse

- **gethostbyname** : résolution DNS

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

- Retourne une structure de type `hostent` pour un nom d'une machine.
 - **name** : le nom de la machine, à récupérer via la commande `hostname`.
 - **hostent** : structure que contient les informations sur la machine.
 - **h_addr_list** : un tableau des adresses de la machine.

```
host=gethostbyname("zellous-MBP");
serveraddr.sin_addr=*(struct in_addr *)host->h_addr_list[0];
```

Les données d'une adresse

■ gethostbyname : Exemple

```
int i;
struct hostent *he;
struct in_addr **addr_list;

he = gethostbyname("www.google.com");

printf("Le nom officiel est : %s\n", he->h_name);
printf("l'adresse est : %s\n", inet_ntoa(*(struct in_addr*)he-
    >h_addr));
printf("toutes les adresses : ");
addr_list = (struct in_addr **)he->h_addr_list;
for(i = 0; addr_list[i] != NULL; i++) printf("%s ",
    inet_ntoa(*addr_list[i]));
```

Merci
