Université des Sciences et de la Technologie
Houari Boumediene
USTHB

La communication interprocessus

Dr BOUYAKOUB F. M
bouyakoub.f.m@gmail.com

Novembre 2012

Plan du cours

- Notion de communication interprocessus
- Communication par partage de variables et par messages
- Communication interprocessus sous Unix: pipe, socket…

Besoins de communication interprocessus

Relation interprocessus

- Dans une activité parallèle (ou pseudo parallèle), un ensemble de processus s'exécutent en parallèle ➔ Deux types de relations entre cet ensemble de processus:
  1. Ces relations peuvent être ***conflictuel***, qui se présentent lorsque les processus mis en jeu partagent des ressources physiques ou logiques;
  2. Ou des relations de ***coopération*** qui se présentent lorsque les processus participent à un traitement global

     ➔ Ces processus auront besoin de communiquer entre eux pour être synchronisés.

Définition de la communication interprocessus

La communication interprocessus (*InterProcess Communication* ou IPC) regroupe l'ensemble des mécanismes permettant la communication et la synchronisation entre processus concurrents (ou distants).

Comment se fait la communication interprocessus?

- **Communication par partage de variables**
  - La synchronisation doit être prise en charge par les processus eux-mêmes (sémaphores, moniteur…)

- **Communication par échange de messages**
  - La synchronisation est gérée par le système

Principe de la communication par variables

- Les processus partagent un ensemble de données communes qui sont soit en mémoire (variables) soit sur disque (fichier).
- Chaque processus peut accéder en lecture ou en écriture à cet ensemble de données appelé espace de données commun.
- Des problèmes peuvent se poser lorsque plusieurs processus essayent d'accéder en même temps à un espace commun.
- Si deux processus ou plus lisent ou écrivent des données partagées, le résultat dépend de l'ordonnancement des processus ➔ **accès concurrents**

Exemple de communication par variables

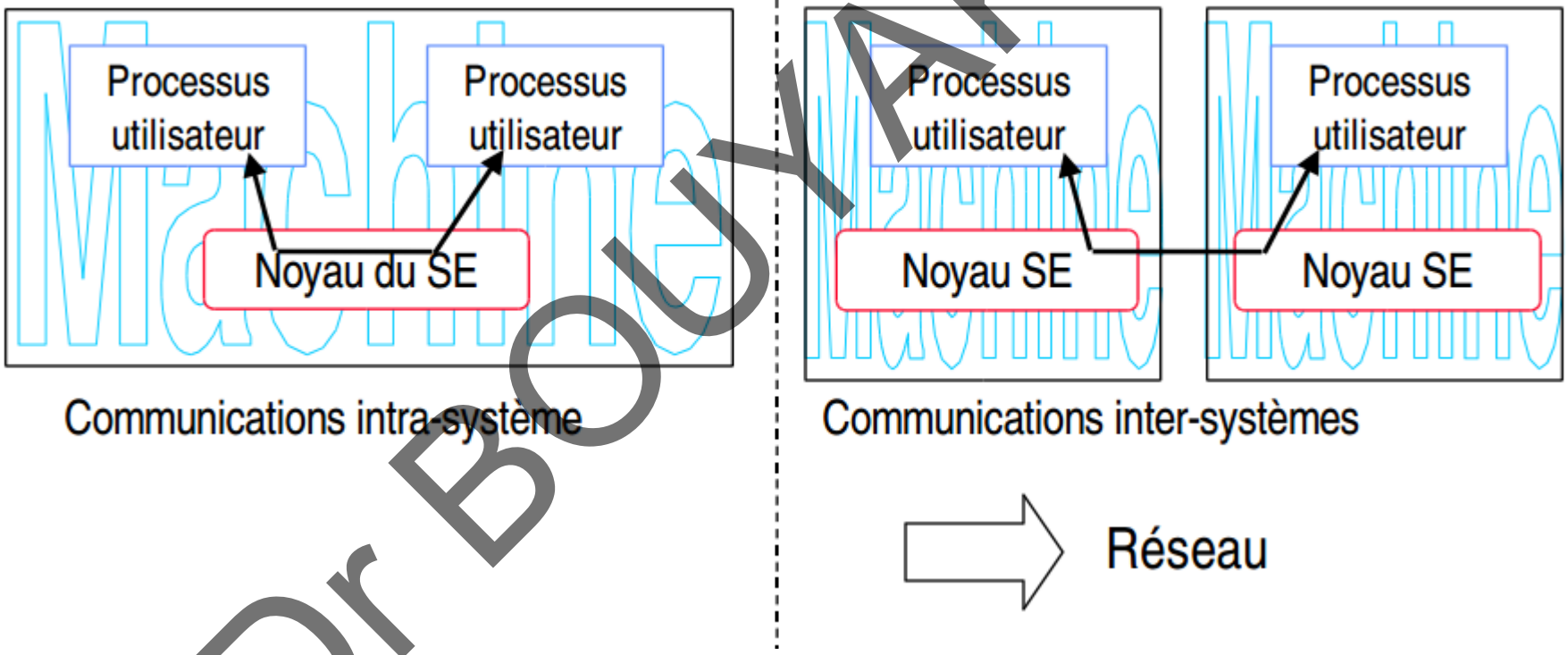
- Producteur(s)/consommateur(s)
- Lecteur(s)/rédacteur(s)

Communication par échange de messages

Définitions

- L'échange de messages permet aux processus de communiquer sans avoir besoin de partager des variables.
- Un message véhicule des données.
- Un processus peut envoyer un message à un autre processus se trouvant sur la même machine ou sur des machines différentes.
- Les liens de communication permettent au SE d'assurer cette communication
- On parle de liaison ***directe***, ***indirecte***, ***unidirectionnelle*** ou ***bidirectionnelle***

Communication intra et inter-systèmes



Communication directe

- Une communication directe: un processus désirant communiquer avec ce mode doit nommer explicitement le destinataire ou l'expéditeur.

- Utilisation de primitives SEND et RECEIVE:
  - SEND (P, message): envoyer « message » à P;
  - RECEIVE(Q, message): Recevoir « message » à Q.

Communication indirecte

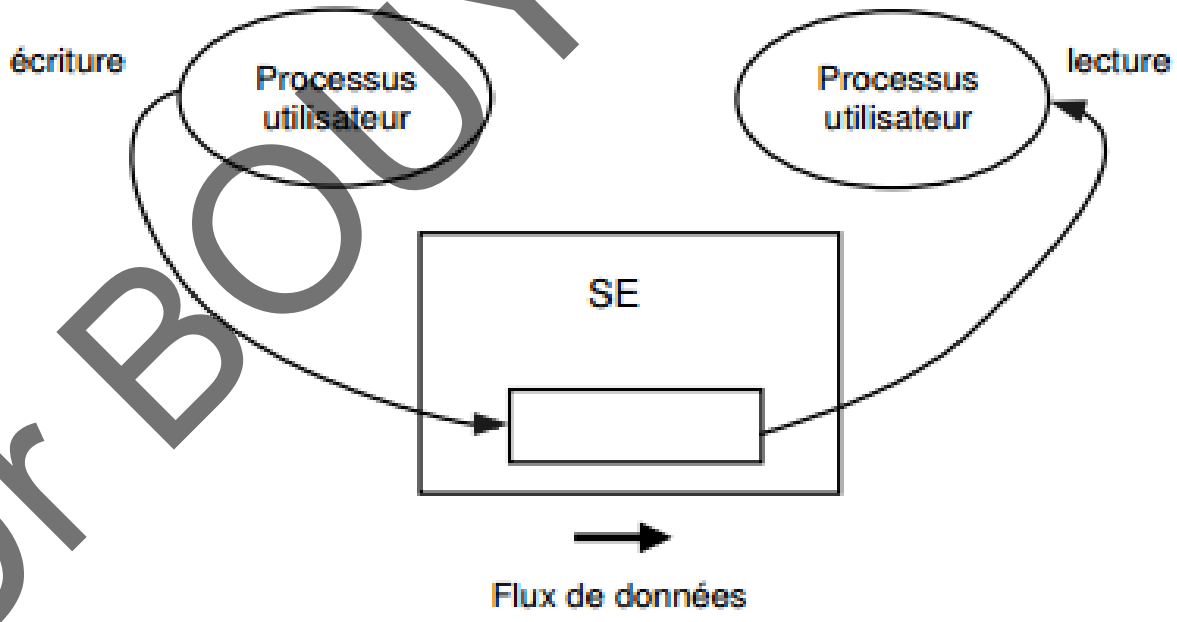
- Dans la communication indirecte les messages sont envoyés et reçus à travers des boîtes aux lettres.
- Une boîtes aux lettres peut être vue comme un emplacement où sont déposés/récupérés les messages échangés.
- Chaque BAL est identifiée d'une façon unique
- Les primitives de communications deviennent:
  - SEND (B, message): envoyer « message » vers la BAL B;
  - RECEIVE(B, message): Recevoir « message » de la BAL B.

Caractéristiques des BAL

- Deux processus ne peuvent communiquer entre eux que s'ils disposent d'une BAL commune
- Deux processus peuvent communiquer entre eux par plus d'une BAL
- Plusieurs processus peuvent utiliser une même BAL pour communiquer entre eux
- Une BAL peut être privée à un processus et dans ce cas il est le seul à l'utiliser en réception
- La BAL disparaît à la fin du processus

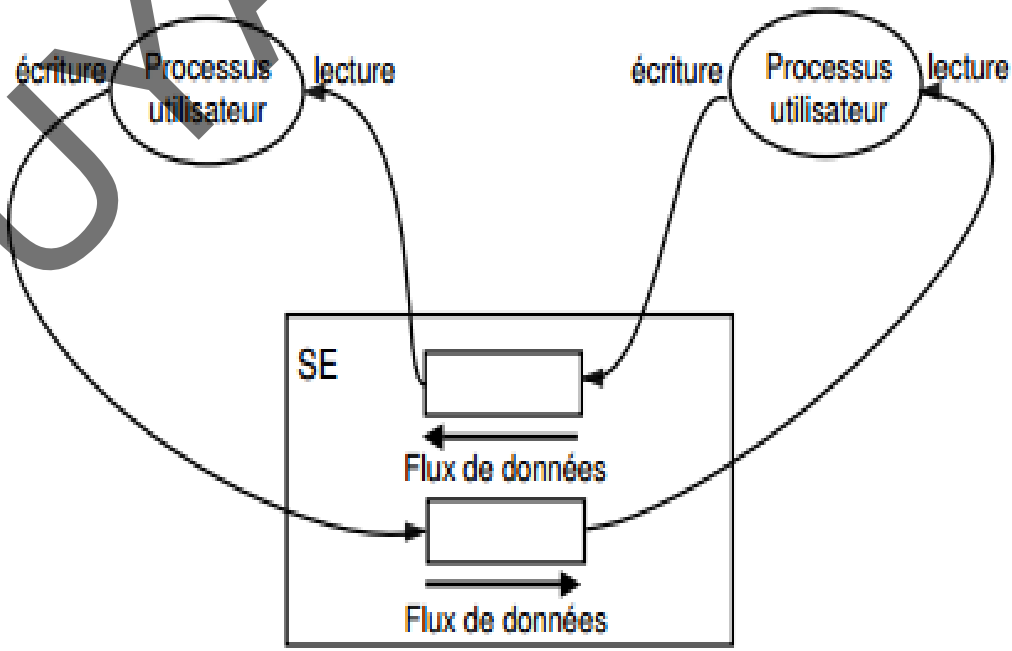
Communication unidirectionnelle

- La communication unidirectionnelle permet d'assurer la communication entre deux processus dans un seul sens



Communication bidirectionnelle

- La communication bidirectionnelle permet d'assurer la communication entre deux processus dans les deux sens.



Capacité des liaisons

- La capacité d'une liaison est le nombre de messages que peut contenir temporairement un lien de communication.
  - **Capacité zéro:** la file a une longueur maximale nulle (pas de messages en attente) ➔ les processus doivent être synchronisés.
  - **Capacité limitée:** la longueur de la file est N (au maximum on peut avoir N messages en attente)➔ si la capacité maximale est atteinte l'émetteur doit attendre jusqu'à ce qu'un message soit récupéré
  - **Capacité illimitée:** la file n'a pas de taille limite

Communication interprocessus sous Unix: les tubes

bouyakoub.f.m@gmail.com

Les tubes ou les pipes

Il existe deux sortes de tube:

- les tubes ordinaires ou non−nommés (volatiles);
- les tubes nommés (persistants).

Définition des tubes

- Un tube est un dispositif qui permet une communication unidirectionnelle.
- Les tubes sont des dispositifs séquentiels ➔ les données sont toujours lues dans l'ordre où elles ont été écrites.
- Un tube est utilisé pour la communication entre deux threads d'un même processus ou entre processus père et fils.

Capacité d'un tube et synchronisation

- La capacité d'un tube est limitée:
  - Si un processus producteur écrit plus vite que la vitesse de consommation des données du processus consommateur ➔ en cas de saturation du tube (capacité gérée par le système) le processus rédacteur est bloqué jusqu'à ce qu'il y ait à nouveau de la place dans le tube.
  - Si le lecteur essaie de lire mais qu'il n'y a plus de données disponibles, il est bloqué jusqu'à ce que ce ne soit plus le cas.

➔ Un tube synchronise automatiquement les deux processus.

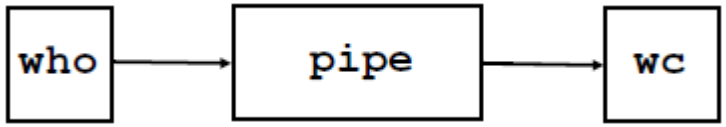
Caractéristiques d'un tube

- Deux descripteurs de fichiers (lecture et écriture);
- Taille limitée (nombre d'adresses directes que contient un nœud du SGF voir chapitre SGF);
- Deux extrémités, permettant chacune soit de lire dans le tube, soit d'y écrire;
- L'opération de lecture dans un tube est destructrice: une information ne peut être lue qu'une seule fois dans un tube;

Création de tube par le shell

- Les tubes sans nom du shell sont créés par l'opérateur binaire «|» qui dirige la sortie standard d'un processus vers l'entrée standard d'un autre processus.

Exemple **$ who | wc -l**



  - Le processus réalisant la commande who ajoute dans le tube une ligne d'information par utilisateur du système.
  - Le processus réalisant la commande wc -l récupère ces lignes d'information pour en calculer le nombre total.
  - Le résultat est affiché à l'écran.

Création de tube avec pipe( )

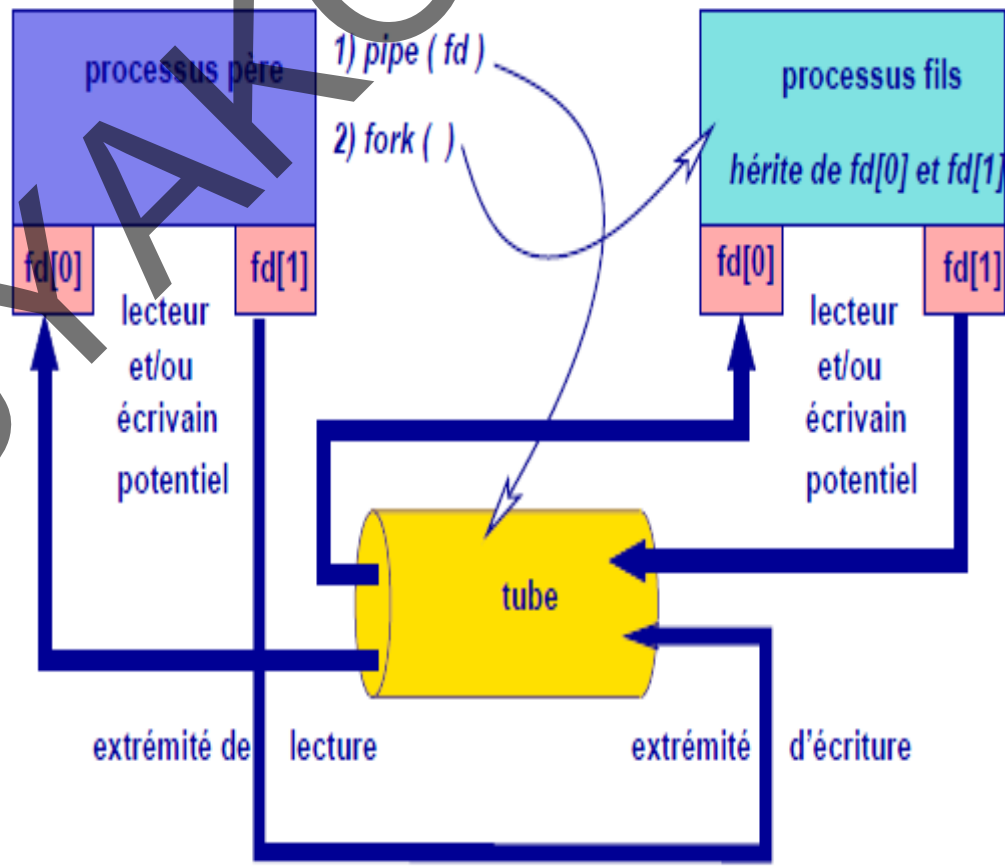
- La création d'un tube se fait grâce à la fonction pipe( ). Elle admet comme paramètre un tableau de deux entiers.

int pipe(int descripteur[2]);

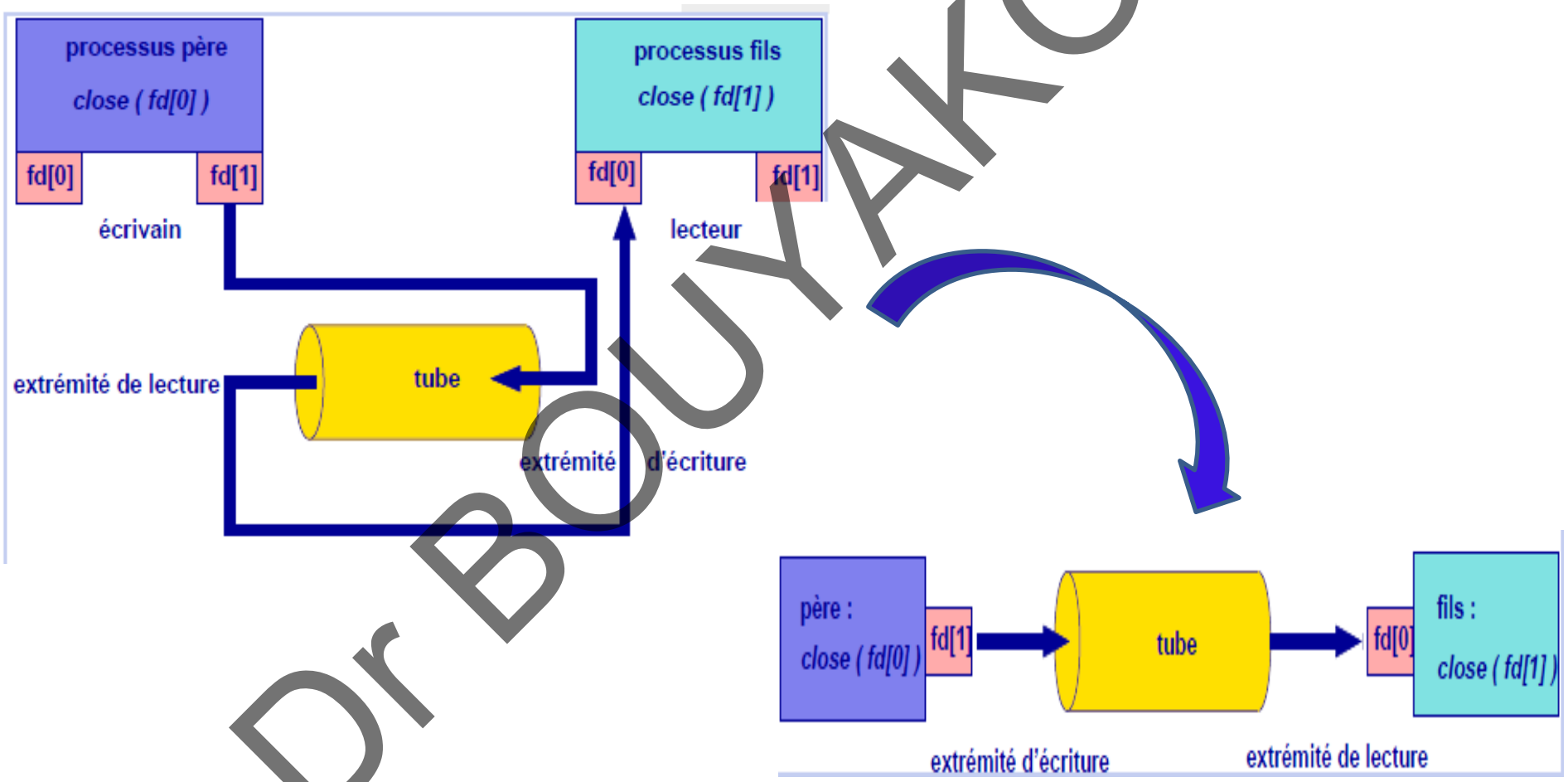
- Au retour de l'appel système pipe( ), un tube sera créé, et les deux positions du tableau passées en paramètre contiennent deux descripteurs de fichiers.
- Un descripteur est considéré comme une valeur entière que le SE utilise pour accéder à un fichier.
- Le descripteur de l'accès en lecture se trouve à la position 0 (**descripteur[0]**)du tableau passé en paramètre, et le descripteur de l'accès en écriture se trouve à la position 1 (**descripteur[1]**).
- Si le système ne peut pas créer de tube, l'appel système pipe() retourne la valeur -1, sinon il retourne la valeur 0.
- Seul le processus créateur du tube et ses fils peuvent accéder au tube.

Effet de création de tube et héritage

```
# include <unistd.h>
main()
{
    ¦
int fd[2];
pipe (fd);
/*création d'un tube avec un descripteur de lecture fd[0] et d'écriture fd[1]*/
/*création d'un fils qui hérite du tube*/
fork( );
    ¦
}
```

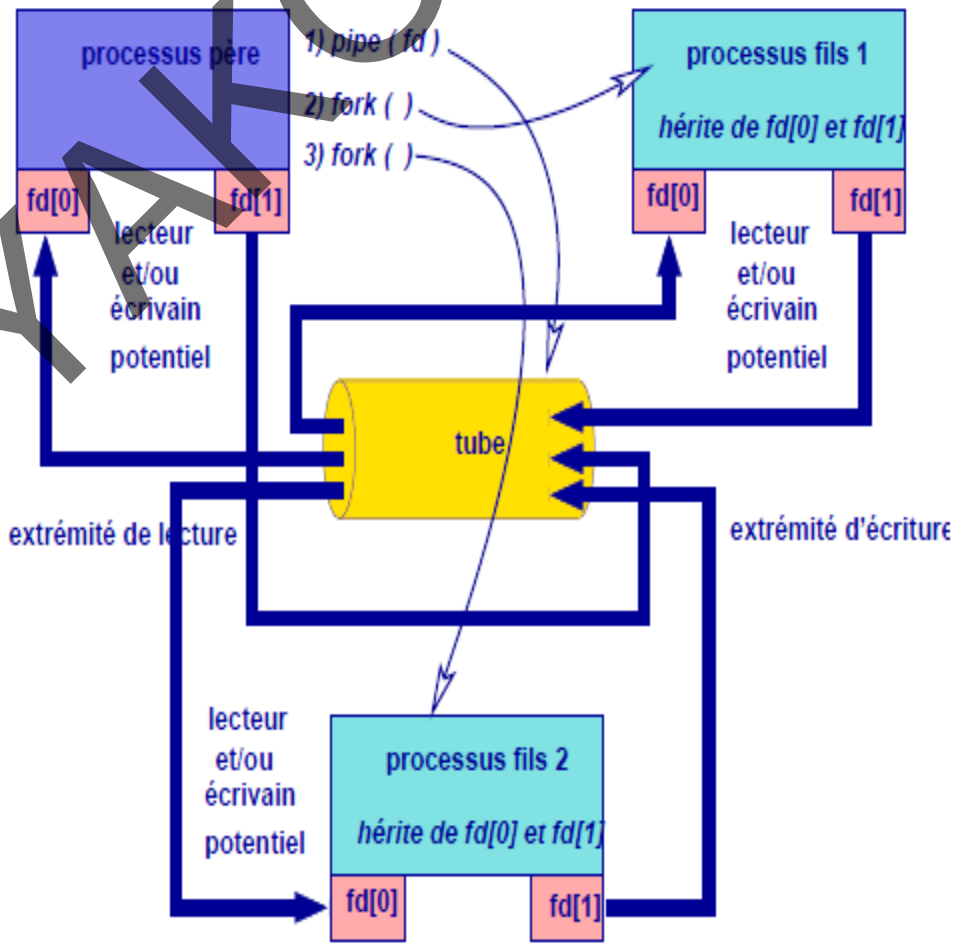


Fermeture des descripteurs non utilisés

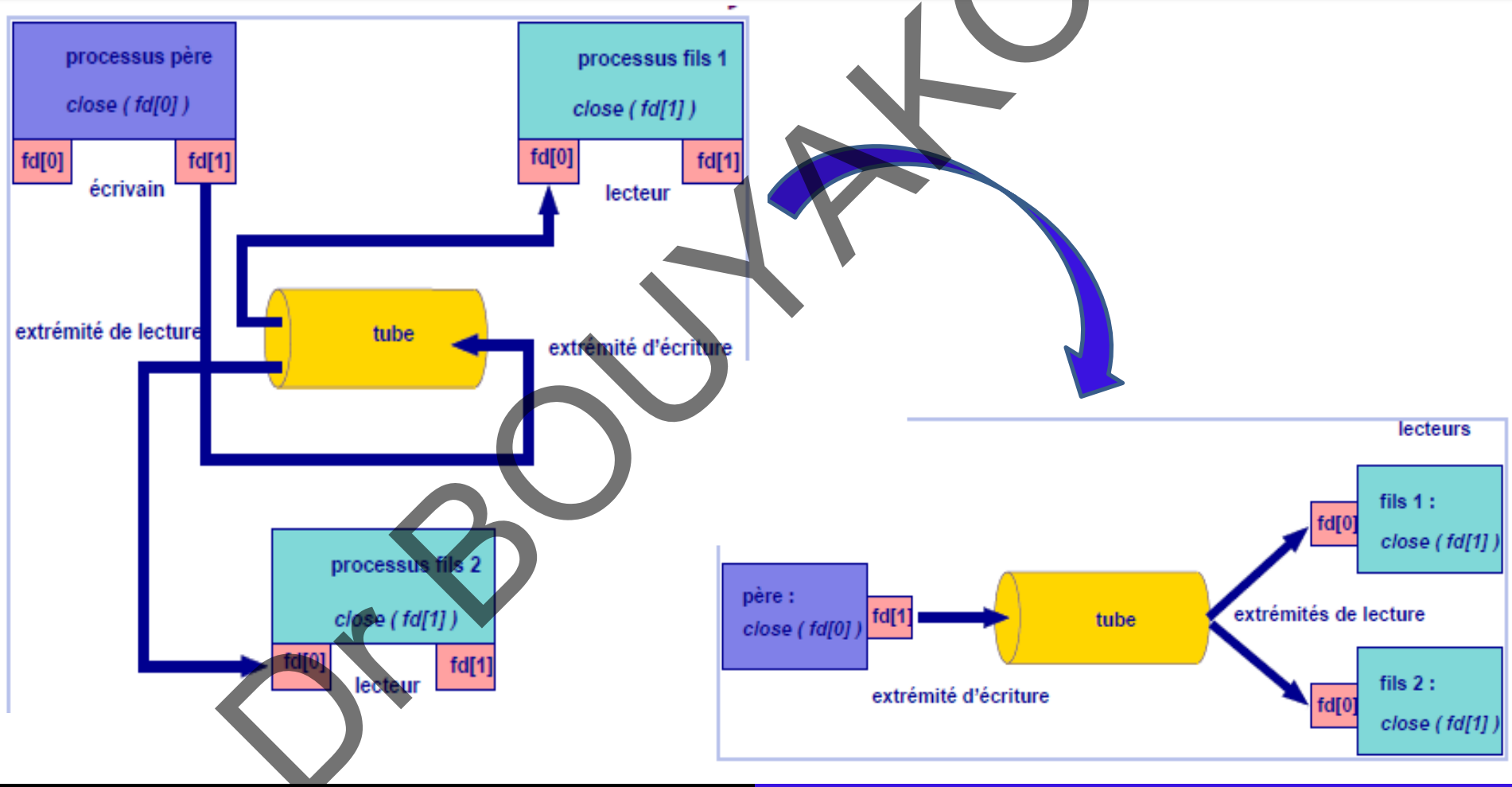


Exemple de création de tube avec héritage de deux fils

```
# include <unistd.h>
main()
{
    ¦
int fd[2];
pipe (fd);
if (fork( ))
fork();
    ¦
}
```



Fermeture des descripteurs pour le père et les fils



La lecture dans le tube

- La primitive de lecture dans un tube p est:

NbLu = read (p[0], buf, nb)

- L'opération de lecture répond à la sémantique suivante:
  - si le tube n'est pas vide, et contient N caractères, le nombre de caractères effectivement lus, NbLu est égal à min (N, nb).
  - Si le tube est vide et que le nombre d'écrivains sur le tube est nul, la fin de fichier est atteinte ➔ NbLu= 0.
  - Si le tube est vide et que le nombre d'écrivains sur le tube n'est pas nul, le processus est bloqué jusqu'à ce que le tube ne soit plus vide.

L'écriture sur un tube

- La primitive d'écriture sur un tube p est:

NbEcrit = write (p[1], buf, nb)

- L'opération d'écriture répond à la sémantique suivante:
  - Si le nombre de lecteurs dans le tube est nul, une erreur est levée par le système et le processus écrivain est terminé.
  - Si le nombre de lecteurs dans le tube n'est pas nul, le retour de la primitive n'a lieu que lorsque les nb caractères ont effectivement été écrits.

Exemple d'utilisation de tube

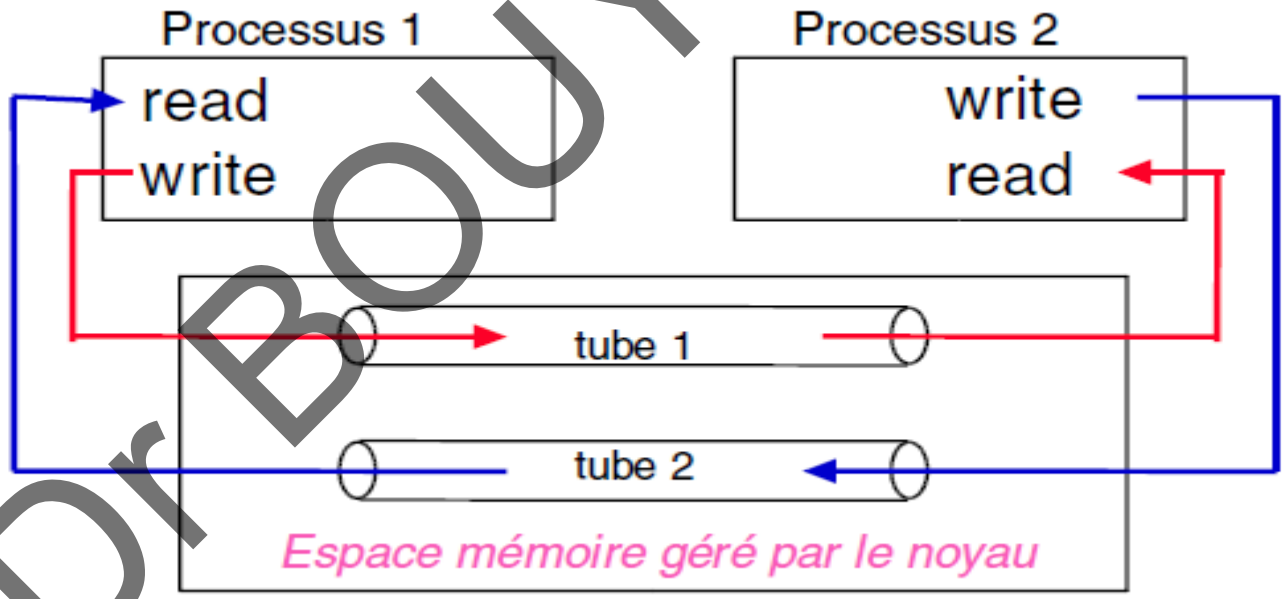
```
# include <unistd.h>
int tube[2];
char buf[20];
main() {
pipe(tube);
  if (fork()) { /* père */
     close (tube[0]);
     write (tube[1], "bonjour", 8);
     close (tube[1]);
     }
  else { /* fils */
     close (tube[1]);
     read (tube[0], buf, 8);
     printf ("%s bien reçu \n", buf);
     close (tube[0]);
    }
}
```

Explication

- Le processus père crée un tube de communication sans nom en utilisant l'appel système **pipe()**;
- Le processus père crée un fils en utilisant l'appel système **fork()**;
- Le processus écrivain (père) ferme l'accès en lecture du tube;
- Le processus lecteur (fils) ferme l'accès en écriture du tube;
- Les processus communiquent en utilisant les appels système **write()** et **read()**;
- Chaque processus ferme son accès au tube lorsqu'il veut mettre fin à la communication.

Communication bidirectionnelles avec les tubes

- La communication bidirectionnelle entre processus est possible en utilisant deux tubes: un pour chaque sens de communication.



Limitations des tubes

- Ils ne permettent qu'une communication unidirectionnelle;

- Les processus pouvant communiquer au moyen d'un tuyau doivent être issus d'un ancêtre commun.

Les tubes nommés ou persistants

- Les tubes nommés fonctionnent comme des files FIFO;
- Les avantages:
  - Ils ont chacun un nom qui existe dans le système de fichiers;
  - Ils sont considérés comme des fichiers spéciaux.
  - Ils peuvent être utilisés par des processus indépendants, à condition qu'ils s'exécutent sur une même machine ➔ ***Deux processus sans relation parent-enfant peuvent échanger des données au moyen des tubes nommés.***

Communication interprocessus sous Unix: les sockets

bouyakoub.f.m@gmail.com

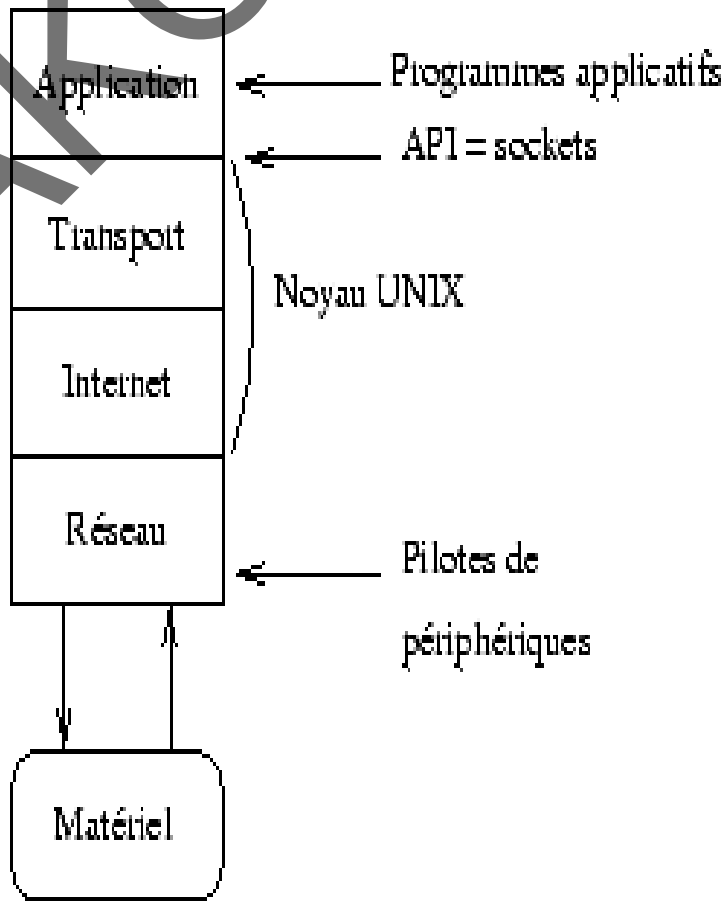
Définition

- C'est un moyen de communication bidirectionnelle pouvant être utilisé pour communiquer avec un autre processus sur la même machine ou avec un processus s'exécutant sur d'autres machines.

- Les applications web communiquent à l'aide de sockets.

Notion d'API

- Les sockets sont une API (*Application Program Interface*) ➔ jouent le rôle d'une interface entre les programmes d'applications et la couche transport (TCP, UDP…).



Le paradigme du client/serveur

- Paradigme très utilisé au sein des applications réparties
- Un serveur:
  - processus rendant un service spécifique identifié par un port particulier (n° port),
  - Le serveur est en attente sur une station (@IP).
- Des clients:
  - processus appelant le serveur afin d'obtenir le service,
  - lancé à la demande à partir généralement de n'importe quelle station.

Le mode connecté et non connecté

- La communication en mode **non-connecté** est une transmission dans laquelle chaque paquet (datagramme) est préfixé par un entête contenant une adresse de destination➔ pas d'établissement de connexion
- Dans une communication en **mode connecté**, les stations qui sont prêtes à échanger des données doivent d'abord se déclarer comme voulant effectivement le faire ➔ Etablissement d'une connexion.

Information nécessaires pour établir une communication

- Afin d'assurer une communication entre deux processus distants on a besoin des informations suivantes (quintuplé):

- protocole,
- port local,
- adresse locale,
- port éloigné,
- adresse éloignée.

Création de socket

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

int socket(int domaine, int type, int protocole);

- Valeur de retour:
  - En cas de réussite ➔ Numéro du descripteur de socket dans la table des descripteurs du processus.
  - En cas d'échec ➔ -1
- Cette primitive nous donne le premier élément du quintuplé:

  {**protocole**, port local, adresse locale, port éloigné, adresse éloignée}

Paramètres de la primitive socket( )

- Domaine d'adresse: Spécifie la famille de protocole à utiliser avec socket (PF Protocol Family ou AF Address Family)
  - Internet: AF_INET;
  - Mode local: AF_LOCAL;
  - Unix: AF_UNIX (idem que AF_LOCAL).
- Type: Spécifie le type de communication désiré.
  - Mode non connecté: SOCK_DGRAM;
  - Mode connecté: SOCK_STREAM.
- Protocole: Permet de spécifier le protocole à utiliser
  - par défaut à 0 (le système choisit le protocole)
  - IPPROTO_UDP pour UDP avec les sockets de type SOCK_DGRAM
  - IPPROTO_TCP pour TCP avec une socket de type SOCK_STREAM

Attachement d'une socket avec la primitive bind()

- Après la création de socket on doit lui associer une @IP et un numéro de port pour pouvoir être utilisée ➔ utiliser la primitive bind()

```
int bind(int  sockfd, struct sockaddr  *myaddr, int  addrlen);
```

- Valeur de retour:
  - retourne 0 si par d'erreur,
  - -1 si erreur.

- Cet appel système complète l'adresse locale et le numéro de port du quintuplé:

  {**protocole, port local, adresse locale**, port éloigné, adresse éloignée}

Paramètres de la primitive bind()

- Paramètres de bind():
  - **Sockfd:** Descripteur de socket (renvoyé par socket()).
  - **Myaddr:** Structure qui spécifie l'adresse locale (@IP + numéro port).
  - **Addrlen:** Entier qui donne la longueur de l'adresse (oct).

Etablissement de la connexion avec connect()

- C'est le processus client qui doit prendre l'initiative de la demande de connexion:

  int connect(int sockfd, struct sockaddr *servaddr, int addrlen) ;

- Les valeurs de retour:
  - -1 dans le cas d'une erreur,
  - 0 dans le cas où la liaison est établie.
- Pour le mode connecté, cet appel système rend la main au code utilisateur une fois que la liaison entre les deux systèmes est établie

  ➔ Tant que cette liaison n'est pas définie, la primitive connect() ne rend pas la main.
- Dans le cas d'un client en mode non connecté un appel à connect() n'est pas faux mais il ne sert à rien et redonne aussitôt la main au code utilisateur.
- Le quintuplé est maintenant complet:

  {**protocole, port local, adresse locale, port éloigné, adresse éloignée**}

Paramètres de la primitive connect()

- **Sockfd:** Descripteur de socket renvoyé par la primitive socket().
- **Servaddr:** Structure qui définit l'adresse du destinataire, la même structure que pour bind().
- **Addrlen:** Donne la longueur de l'adresse, en octets.

Accepter une connexion avec accept()

- Cette primitive permet d'accepter les connexions entrantes ➔ Le serveur se met en attente bloquante d'une connexion:

  int accept(int sockfd, void *addr, int *addrlen);

- Les paramètres d'accept():
  - **sockfd:** descripteur de socket,
  - **addr:** Pointeur sur une structure de type sockaddr,
  - **addlen:** longueur de addr.

Utilisation de la primitive accept()

newsock = accept(sockfd, addr, addrlen) ;

- Quand une connexion arrive, on remplit la structure addr avec l'adresse du client qui fait la demande de connexion.
- Puis le système crée une nouvelle socket dont il renvoie le descripteur (récupéré ici dans newsock)
  ➔ La socket originale reste disponible pour d'autres connexions.

Envoyer des données

- Une fois qu'un programme d'application a créé une socket, il peut l'utiliser pour transmettre des données via: send() et sendto().
- Send() fonctionne uniquement en mode connecté, ➔ elle n'offre pas la possibilité de préciser l'adresse du destinataire.
- Sendto(): permet d'envoyer un message en mode non connecté ➔ On doit spécifier l'adresse du destinataire à chaque appel.

La primitive send()

int send(int s, void *msg, int len, int flags) ;

- Les paramètres de send():
  - **S:** le descripteur de socket,
  - **msg:** L'adresse du début de la suite d'octets à transmettre;
  - **len:** Spécifie le nombre de ces octets,
  - **flags:** Permet de paramétrer la transmission du data-gramme, notamment si le buffer d'écriture est plein (par défaut à 0).

La primitive sendto()

int sendto(int s, void *msg, int len, int flags, void *to, int tolen);

- Les paramètres de sendto():
  - Les quatre premiers arguments sont exactement les mêmes que pour send(),
  - les deux derniers permettent de spécifier une adresse et sa longueur avec une structure du type sockaddr, (comme vu précédemment avec bind()).

Recevoir des données

- Deux primitives pour lire des messages sur le réseau:

```
int recv(int s, void *buf, int len, int flags) ;
```

  - **s:** descripteur de socket,
  - **buf:** Adresse où l'on peut écrire en mémoire,
  - **len:** Longueur du buffer,
  - **flags:** Permet au lecteur d'effectuer un contrôle sur les paquets lus.

```
int recvfrom(int s, void *buf, int len, int flags, void *from, int *fromlen);
```

  - Les deux arguments additionnels par rapport à recv() sont des pointeurs vers une structure de type sockaddr et sa longueur.

File d'attente de connexion

- Pendant la communication avec un client, le serveur peut recevoir une requête d'un autre client ➔le serveur étant occupé, la requête n'est pas prise en compte et le système refuse la connexion.
- La primitive listen() permet la mise en file d'attente des demandes de connexions (utilisée après socket() et bind() et avant accept()).

```
int listen(int sockfd, int backlog);
```

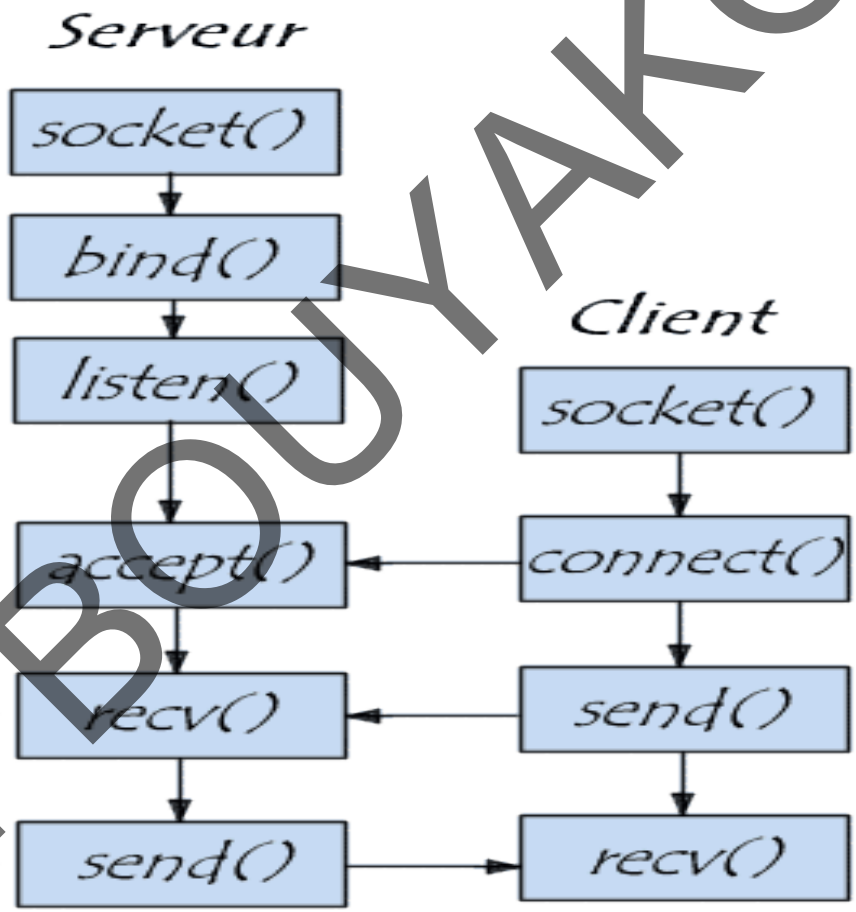
- **sockfd:** descripteur de socket,
- **backlog:** Nombre de connexions possibles en attente (de 1 à 32).

Fermeture de socket

- A la fin de la communication, le processus doit fermer la socket avec la primitive close().

close (*descripteur de socket*);

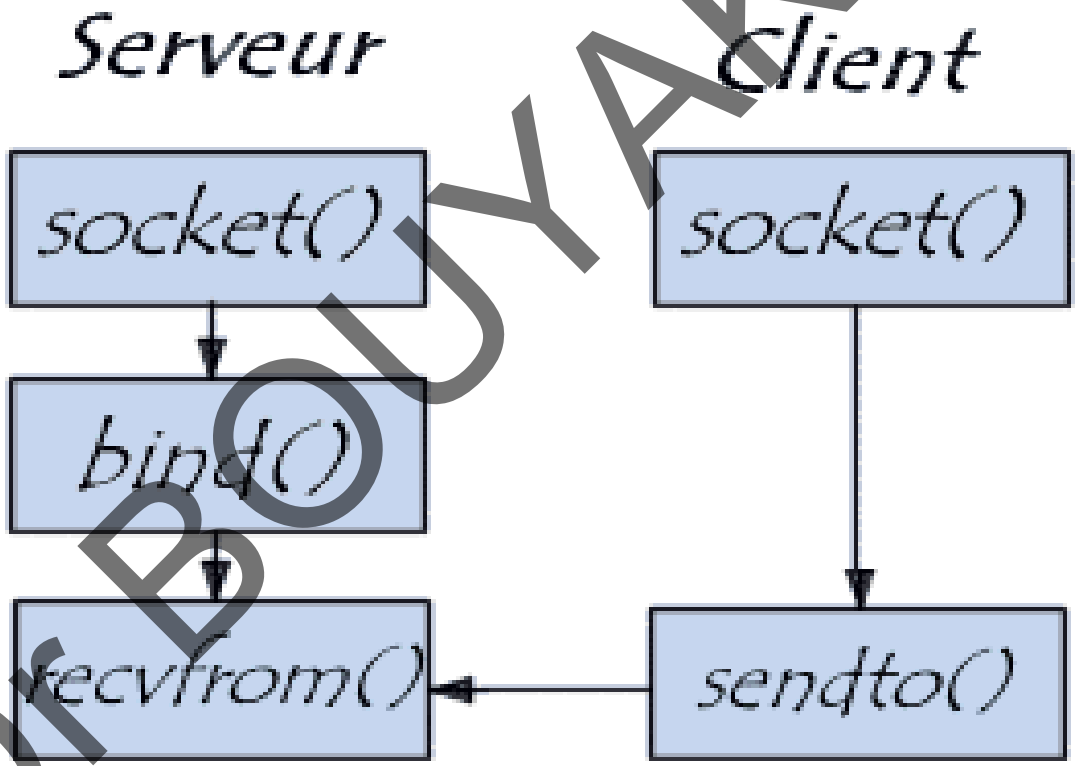
Scénario d'exécution en mode connecté



Interprétation du scénario

- Côté Client (demandeur de la connexion)
  - Crée une socket socket()
  - Se connecte à une <adresse,port> connect()
  - Lit et écrit dans la socket send(), recv()
  - Ferme la socket close()
- Côté Serveur (en attente de connexion)
  - Crée une socket socket()
  - Associe une adresse au socket bind()
  - Se met à l'écoute des connexions entrantes listen()
  - Accepte une connexion entrante accept()
  - Lit et écrit sur la socket recv(), send()
  - Ferme la socket close()

Scénario d'exécution en mode non connecté



Interprétation du scénario

- Côté client
  - Crée une socket socket()
  - Envoi d'un message dans la socket sendto()
  - Libère la socket close()
- Côté serveur
  - Crée une socket socket()
  - Associe une adresse au socket bind()
  - Ecoute et réception d 'un message recvfrom()
  - Libère la socket close()

Questions ?