Cours

Systèmes d'information distribués

Dr. Abderrahmane Baadache

abderrahmane.baadache@gmail.com

Mars, 2021

1. Introduction sur les systèmes d'information

a. Définition

- **Un SI (Système d'information)** est un ensemble organisé de ressources (données, procédures, matériel, logiciel, etc.) permettant d'acquérir, de stocker, de structurer et de communiquer des informations sous forme de textes, images, sons, ou de données au sein des organisations.
- On distingue des systèmes d'information :
  - **Supports d'opérations** (traitement de transaction, contrôle de processus industriels, supports d'opérations de bureau et de communication)
  - **Supports de gestion** (aide à la production de rapports, aide à la décision, etc.).
- **Exemple** : Système d'information médical.
  - Bases de données (dossiers médicaux + l'administratif), infrastructure réseau.
  - Applications médicales de gestion : administrative, médicaments, équipements, examens, planification, etc.

b. Fonctions d'un SI

- **Échange de données** entre applications hétérogènes manipulant des données au format hétérogènes.
- **Répartition** (processus et données) sur des sites géographiques distants.
- **Interopérabilité** des plates-formes.
- **Portabilité** des applications.
- Gestion de la **réplication** : Serveurs de fichiers répliqués, bases de données répliquées, Site Web miroirs.
- Gestion de la **cohérence des données**.
- Gestion des **accès concurrents**.
- **Persistance** des données.
- **Tolérance aux pannes**.
- **Scalabilité** : performance garantie si nombre de sites, distance entre les sites, nombre d'utilisateurs, nombre de processus, volume des données et fréquence des interactions augmentent.
- **Sécurité** (Authentification, intégrité, confidentialité, anonymat, contrôle d'accès)

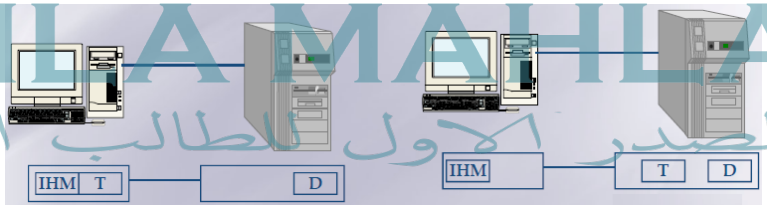
c. Systèmes centralisés

- Composants localisés sur un site unique.
  - **1 processeur** : une horloge commune.
  - **1 mémoire centrale** : un espace d'adressage commun.
  - **1 système d'exploitation** : gestion centralisée des ressources et un etat global facilement reconstituable.
  - **Accès local aux ressources**
- Centralisation des données et des traitements.
- Presque abandonné car n'est pas pratique.

d. Client-serveur

Client-serveur désigne une interaction à travers un réseau entre plusieurs processus, l'un qualifié de **client**, envoie des requêtes. L'autre, qualifié de **serveur**, attend les requêtes des clients et y répond.

- **2 niveaux (2-tier)**



⇒ Architecture dénommée `client lourd`

- Le poste de travail héberge l'ensemble de la gestion d'interface homme-machine et le traitement.
- Le serveur est un serveur de base de données.

⇒ Architecture dénommée `client léger`

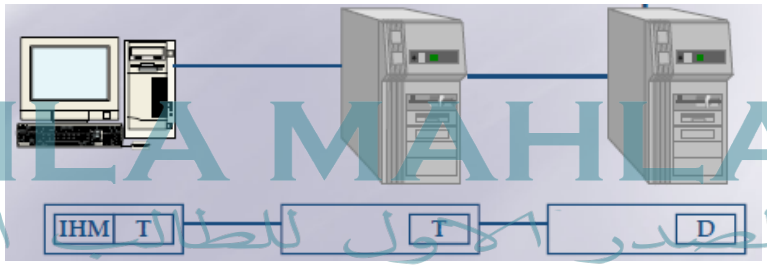
- Le poste de travail n'héberge que l'interface homme-machine.

- Le serveur héberge les données et les traitements.

Avantages & inconvénients

+ Mise en œuvre relativement simple.
+ Efficace pour un nombre réduit de clients.
- Coûts de déploiement et de MAJ.
- Accès concurrent limité.

- **3 niveaux (3-tier)**



- Le poste de travail héberge l'interface homme-machine et une partie des traitements.
- Le serveur d'applications gère l'autre partie des traitements.
- Le serveur de données gère les accès aux données.
- Architecture dénommée `traitements coopératifs`.

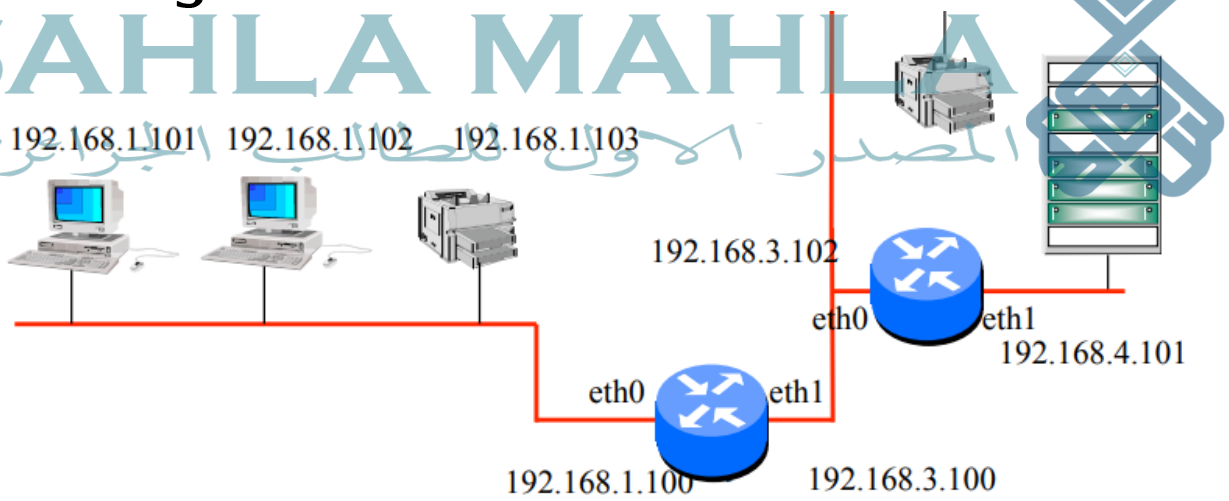
Avantages

- Centralisation des traitements.
- Pas de duplication des données.
- Gestion simple de la cohérence des données.

e. Quelques rappels

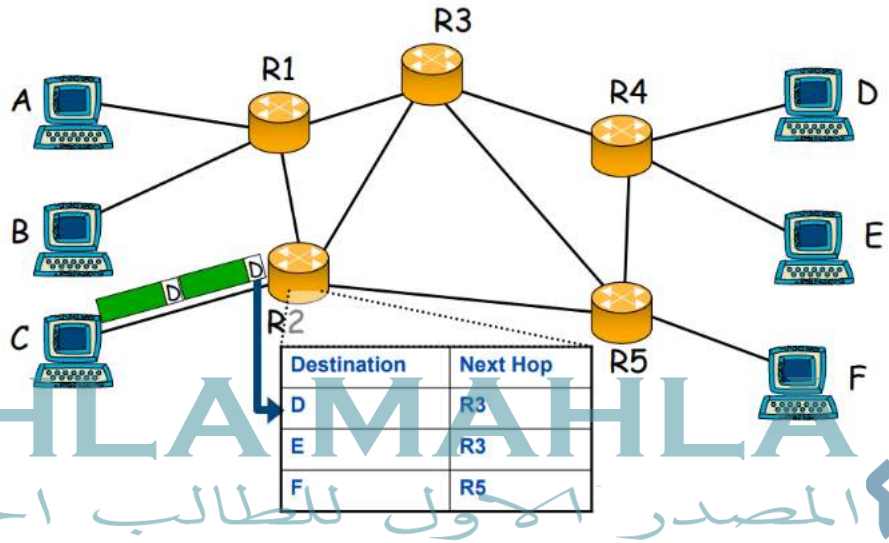
- Une **adresse IP** est un quadruplet de nombres de 8 bits chaque, séparés par des points. Exemple : 192.168.1.10
- **@IP 127.0.0.1** : adresse de rebouclage (ang. loopback) qui désigne la machine locale (ang. localhost).
- **5 classes**
  - **Classe A** : de 1.0.0.1 à 126.255.255.254. Le 1er octet réseau (NetID), les 3 octets suivants pour l'@ de l'hôte (hostID))
  - **Classe B** : de 128.0.0.1 à 191.255.255.254. Les deux 1ers octets réseau, 2 octets suivants pour l'@ de l'hôte.
  - **Classe C** : de 192.0.0.1 à 223.255.255.254. Les trois 1ers octets réseau, 4éme octet pour l'@ de l'hôte
  - **Classe D** : de 224.0.0.1 à 239.255.255.255. Groupes multicast.
  - **Classe E** : réservées pour futur usage.
- **IPv6** résout le problème de pénurie d'adresses IP. @IP sur 128 bits.

- **Port d'écoute**
  - 16-bits unsigned integer, de 1 à 65535.
  - Ports utilisateur >= 1024.
  - Exemples de ports réservés : FTP : 21/TCP, Telnet : 23/TCP, SMTP : 25/TCP, Login : 513/TCP, HTTP : 80/TCP.
- **Routage dans les réseaux IP**



  - Le **routage** est l'opération qui consiste à trouver un chemin entre deux machines, l'une émettrice et l'autre réceptrice.
  - Un routeur est un matériel réseau spécifique, conçu spécialement pour le routage.

– Les routeurs se réfèrent à des **tables de routage** internes pour prendre des décisions d'acheminement des paquets le long des chemins du réseau.



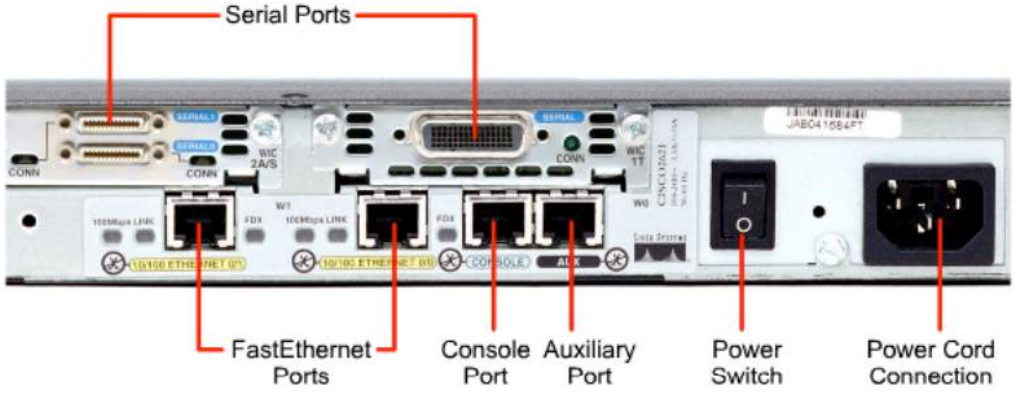
– Le routage peut être **statique** ou **dynamique**
  ⋆ En statique, un administrateur réseau établit manuellement des tables de routage statiques.
  ⋆ Les tables de routage dynamiques se mettent à jour automatiquement.
– Quelques protocoles de routage
  ⋆ **IP** : l'Internet Protocol (IP) spécifie l'origine et la destination de chaque paquet

de données. Les routeurs inspectent l'entête IP de chaque paquet pour identifier où les envoyer.

⋆ **OSPF** : le protocole OSPF (Open Shortest Path First) est couramment utilisé par les routeurs de réseau pour identifier dynamiquement les routes disponibles les plus rapides et les plus courtes pour envoyer des paquets à leur destination.

⋆ **RIP** : le protocole RIP (Routing Information Protocol) utilise le " comptage de sauts " pour trouver le chemin le plus court d'un réseau à un autre.

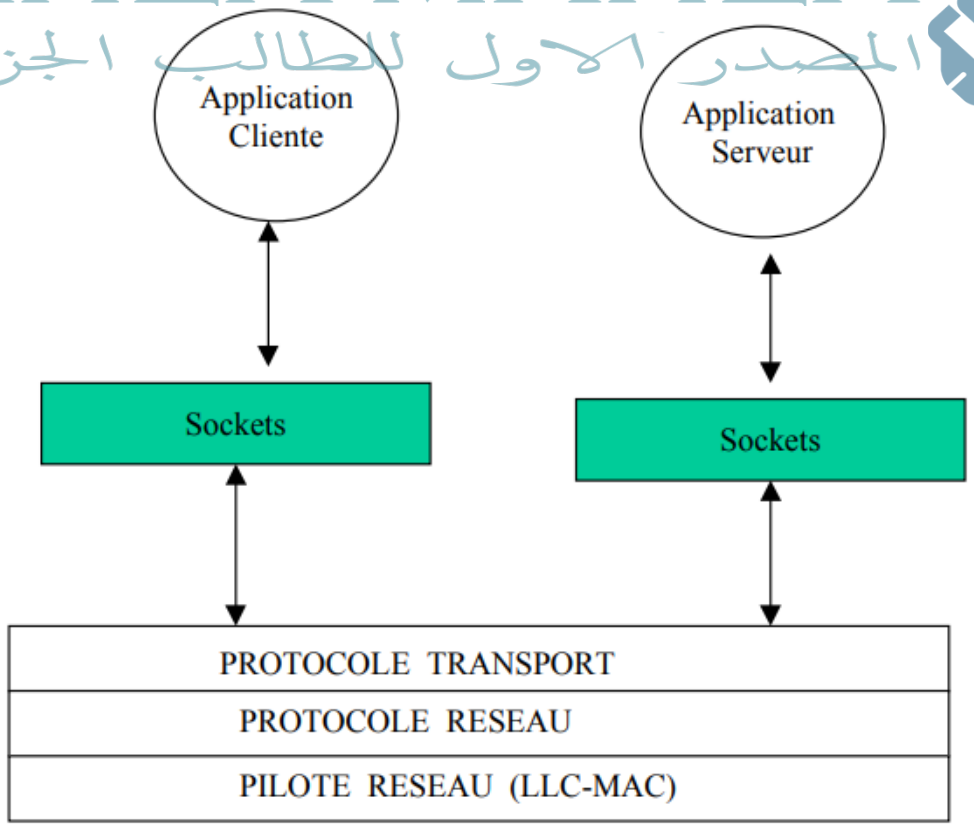
– Connections du **2600 router**



2. Les sockets

a. Notion de sockets

– Il s'agit d'un modèle permettant la communication inter processus (IPC - Inter Process Communication).
– En d'autres termes, une **socket** est un point de communication par lequel un processus peut émettre ou recevoir des données.



2. Les sockets

- Pour établir une communication vers une machine distante, il faut :
  - s'attribuer un **numéro de port**.
  - déterminer l'**adresse Internet de la machine réceptrice**.
  - déterminer le **numéro de port du service distant**.
- Modes de communication
  - Le **mode connecté**, comparable à une communication téléphonique, utilisant le protocole **TCP**, dans lequel une connexion durable est établie entre les processus.
  - Le **mode non connecté**, analogue à une communication par courrier, utilisant le protocole **UDP**. Il s'agit d'une communication sans connexion, rapide mais moins fiable.
- Types de socket
  - Stream Sockets (TCP) : communication en mode connecté.
  - Datagram Sockets (UDP) : communication en mode non connecté.

2. Les sockets

○○○

b. Fonctionnement des sockets

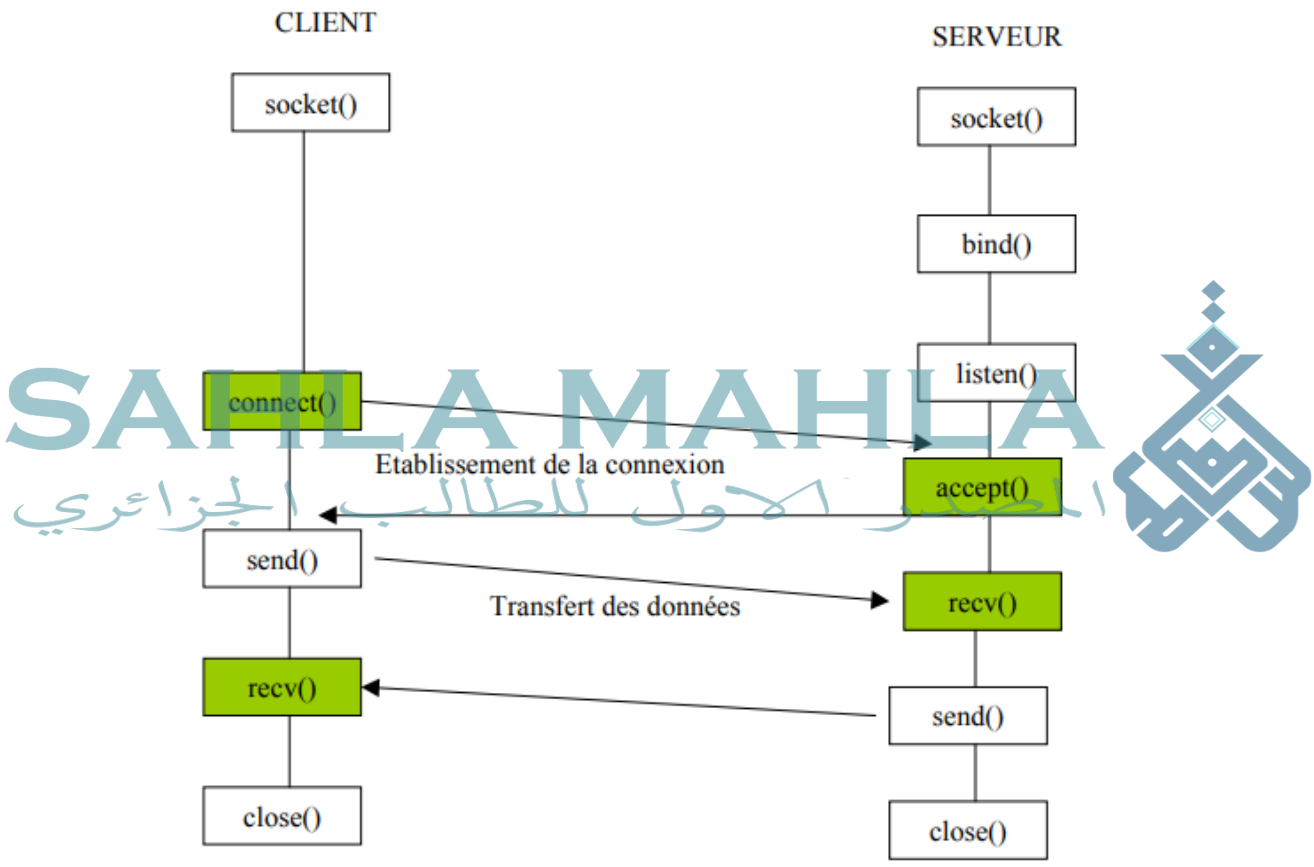
⇒ Coté client (en mode connecté)

1. crée une socket.
2. se connecte au serveur en donnant l'adresse socket distante (adresse Internet du serveur et numéro de port du service).
3. lit ou écrit sur la socket.
4. ferme la socket.

⇒ Coté serveur (en mode connecté)

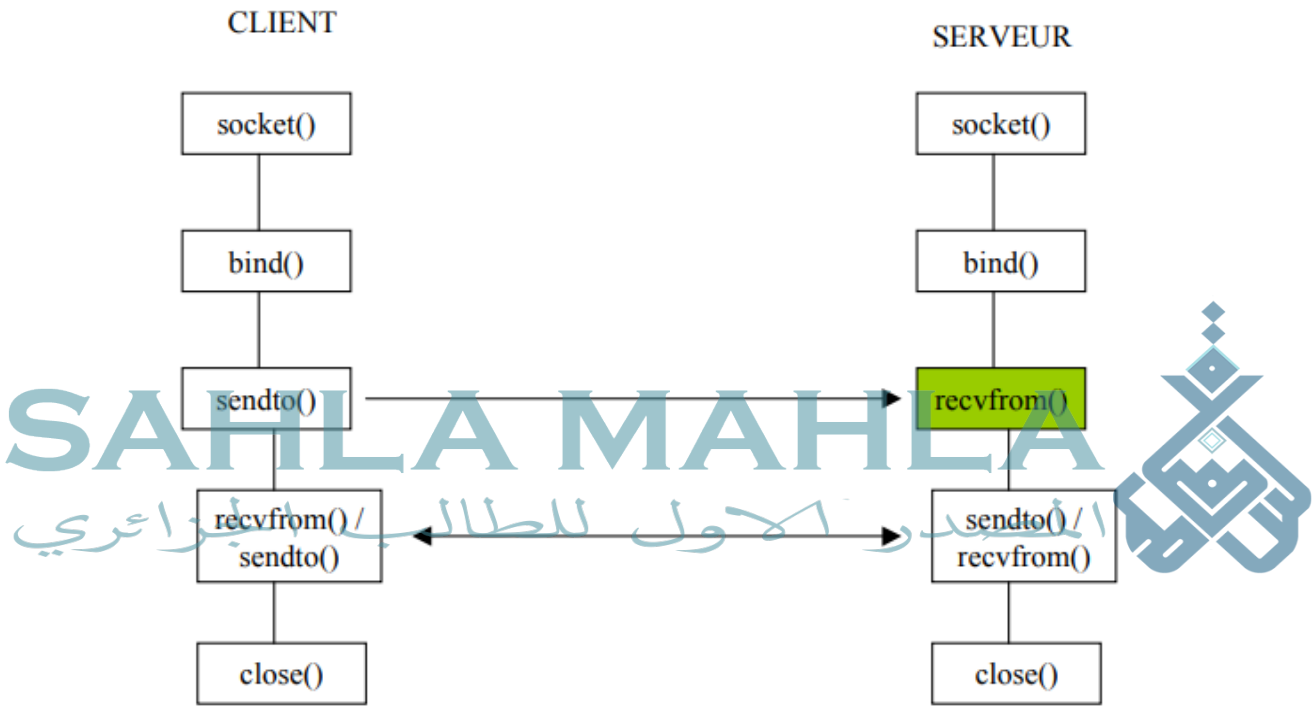
1. crée une socket.
2. associe une adresse socket (son adresse Internet et le numéro de port choisi) au service : binding.
3. se met à l'écoute des connexions entrantes.
4. Pour chaque connexion entrante :
   - accepte la connexion.
   - lit ou écrit sur la nouvelle socket.
   - ferme la nouvelle socket.

⇒ Client/Serveur en mode connecté (TCP)



En mode connecté, une connexion doit être établie avant que la transmission effective des données commence, cette connexion est libérée à la fin de la transmission.

⇒ Client/Serveur en mode non connecté (UDP)



En mode non connecté, aucune connexion n'est exigée.

A noter que l'implémentation des sockets se diffère d'un langage de programmation à un autre et sous un système d'exploitation à un autre.

c. Sockets en C sous Linux

⇒ Création d'une socket

```
int socket (domaine, type, protocole) ;
int domaine ; /* AF_INET ou AF_UNIX */
int type ;/* SOCK_STREAM (TCP) ou SOCK_DGRAM
(UDP)*/
int protocole ; /* IPPROTO_TCP ou IPPROTO_UDP
équivalent si = 0 */
```


⇒ Primitive bind (TCP et UDP)

```
int bind (socket, adresse, l_adresse) ;
int socket ; /* descripteur de la socket */
struct sockaddr *adresse ; /*adresse socket
(port+adresse) */
int l_adresse ; /* longueur de la structure
adresse */
```

⇒ Primitive connect (TCP)

```
int connect (socket, adresse, l_adresse) ;
int socket ; /* descripteur de la socket */
struct sockaddr *adresse ; /*adresse socket
(port+adresse) */
int l_adresse ; /* longueur de la structure
adresse */
```

2. Les sockets

⇒ Primitives listen (TCP)

Mise en écoute : indique que le serveur peut attendre au maximum qlen requêtes des clients.

```
int listen (socket, qlen) ;
int socket ; /* descripteur socket */
int qlen ; /* nombre maximal de connexions
traitées */
```

⇒ Primitive accept (TCP)

```
int accept(socket, struct sockaddr *pin, addrlen)
int socket ; /* descripteur retourné par la
primitive socket */
struct sockaddr *adresse ; /*initialisé à vide
et mis à jour par accept avec le port et l'ad-
resse IP de la machine où se trouve le client
qui a fait un connect */
int l_adresse ;/*longueur de la structure
adresse*/
```

⇒ Primitives envoi & réception en mode TCP

```
ssize_t recv(
int descripteur,
void *ptr,
size_t nb_caracteres,
int option /* = 0, MSG_PEEK pour consulter sans
extraire, MSG_OOB pour lire une donnée urgente
(1 caractère) */
) ;
ssize_t send(
int descripteur,
void *ptr,
size_t nb_caracteres,
int option /* =0 ou MSG_OOB */
) ;
```


⇒ Primitive de réception en mode UDP

```
int recvfrom(
int descripteur,/* descriptor de la socket */
void * message, /* adresse de réception */
int longueur, /* taille zone réservée */
int option, /* 0 ou MSG_PEEK */
```

```
struct sockaddr *ptr_adresse,/*adr émetteur*/
int *ptr_longueur_adresse /* pointeur sur long-
ueur zone adresse */
)
```

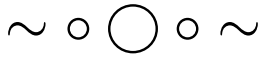
⇒ Primitive d'envoi en mode UDP

```
int sendto(
int descripteur,/*descripteur de la socket */
void * message, /* adresse à envoyer */
int longueur, /* longueur du message */
int option, /* 0*/
struct sockaddr *ptr_adresse, /* adr dest */
int longueur_adresse /*longueur adr dest */
)
```


⇒ Primitive close (TCP et UDP)

Fermeture de connexion

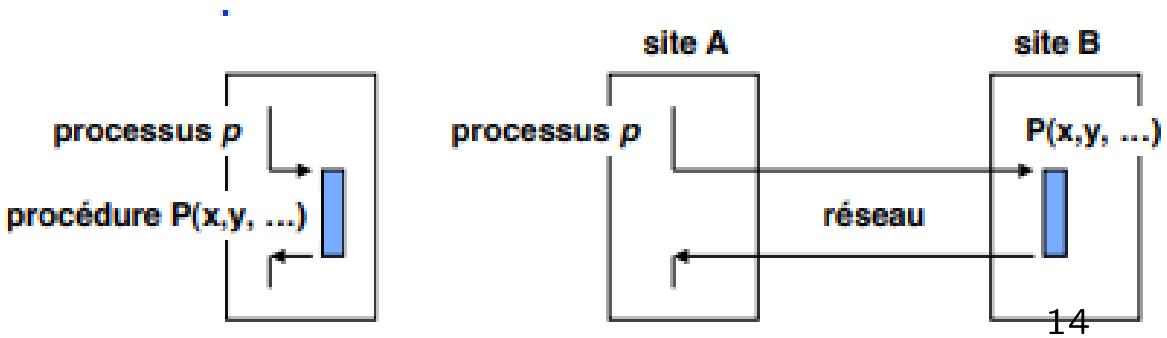
```
int close (socket)
int socket ; /* descripteur de socket */
```



3. Remote Procedure Call (RPC)

a. Appel de procédure distante

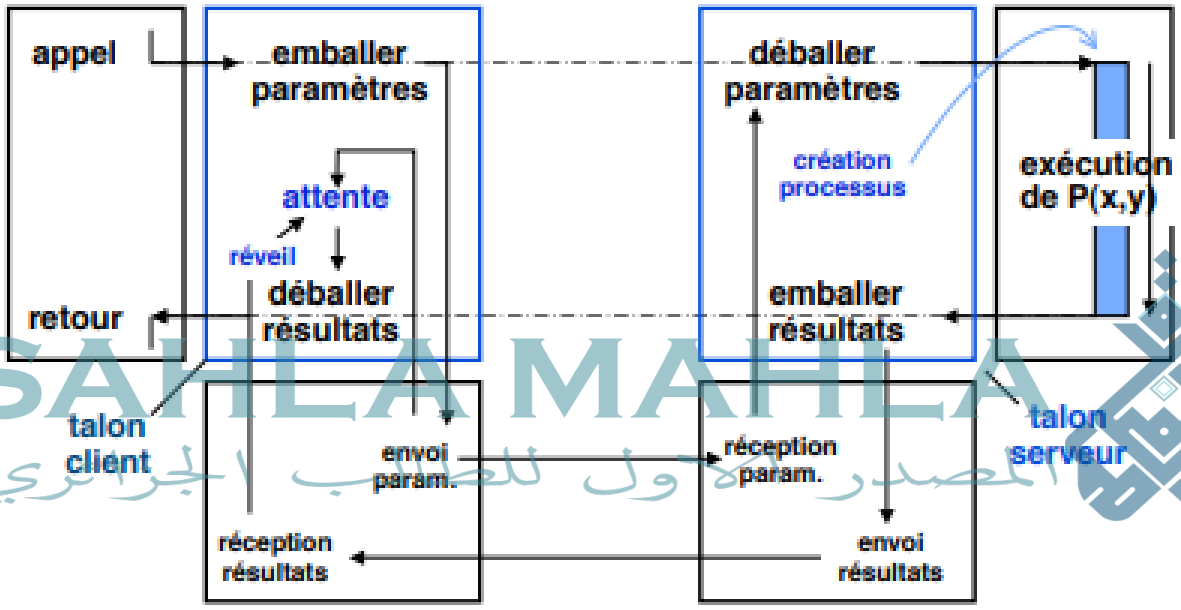
- Appel de procédures locales : **Appelant** et **appelé** dans le même espace virtuel : même mode de pannes, appel et retour fiable.
- Appel de procédures distantes : **Appelant** et **appelé** dans deux espaces virtuels : mode de pannes indépendant, réseau non fiable, temps de réponse.
- **Remote Procedure Call (RPC)** ou appel de procédure distante est une technique de communication inter-processus utilisée pour les applications client-serveur. Il est également appelé *appel de sous-programme* ou *appel de fonction*.



3. Remote Procedure Call (RPC)

○○○

b. Fonctionnement



⇒ Fonctions des talons (Stub et Skeleton)

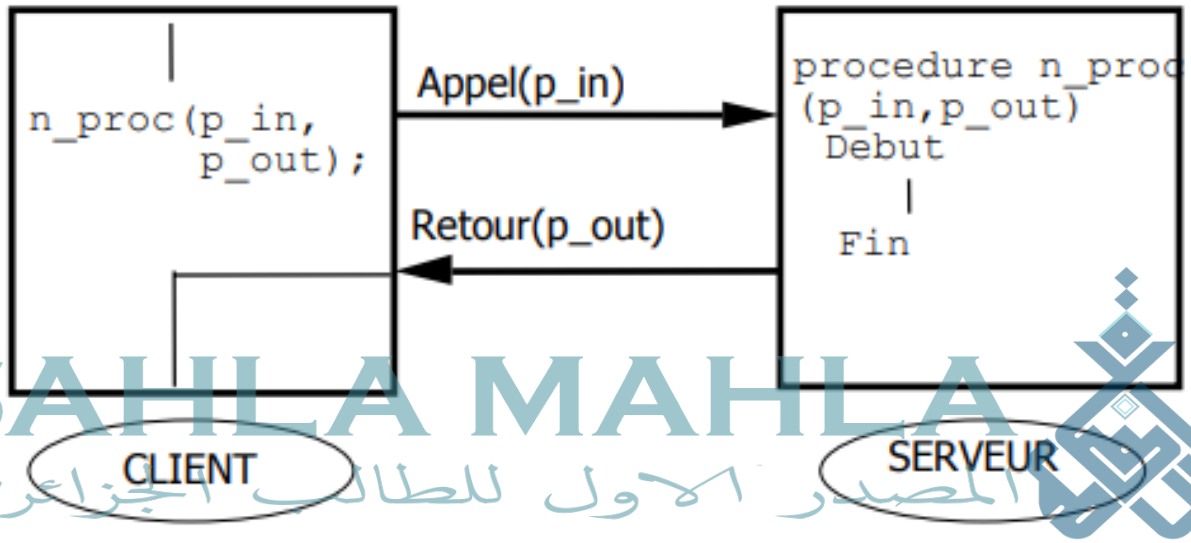
- **Talon client (Stub)**
  - Représente le serveur sur le site client.
  - Reçoit l'appel local.
  - Emballe les paramètres.
  - Crée un identificateur unique pour l'appel.
  - Exécute l'appel distant.

  - Met le proc. client en attente.
  - Reçoit et déballe les résultats.
  - Exécute le retour vers l'appelant (comme retour local de procédure)
- **Talon serveur (Skeleton)**
  - Représente le client sur le site serveur.
  - Reçoit l'appel sous forme de message et déballe les paramètres.
  - Crée ou sélectionne un processus (ou thread) et lui fait exécuter la procédure.
  - Emballe les résultats et les transmet à l'appelant.

⇒ Mise en œuvre de RPC

- **Par migration** : Le code et les données de la procédure distante sont amenés sur le site appelant pour y être exécutés par un appel local habituel.
- **Par mémoire partagée** : L'appel distant est réalisé en utilisant une **mémoire virtuelle partagée répartie**. La procédure est installée pour le client comme pour le serveur dans la mémoire partagée répartie.

- **Par messages asynchrones** : Deux messages (au moins) échangés : requête et réponse.



- **Par appel léger (lightweight RPC)** : la communication réseau est réalisée par un segment de mémoire partagée entre le client et le serveur qui contient un tas pour les paramètres d'appel et de réponse.

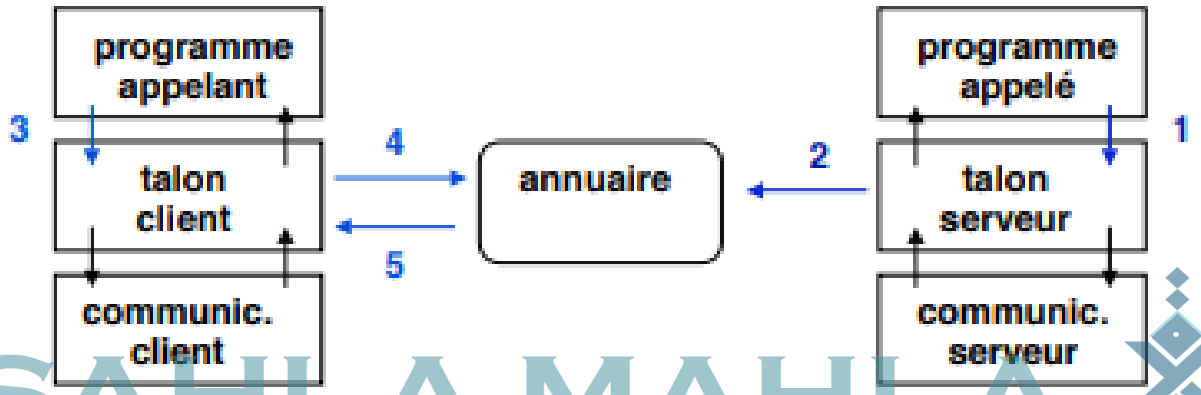
c. Désignation et liaison

– Comment sont structurés les noms et références permettant de désigner les services distants ?
– Comment localiser une procédure distante ?

3. Remote Procedure Call (RPC))

○○○

⇒ Désignation et liaison utilisant un annuaire

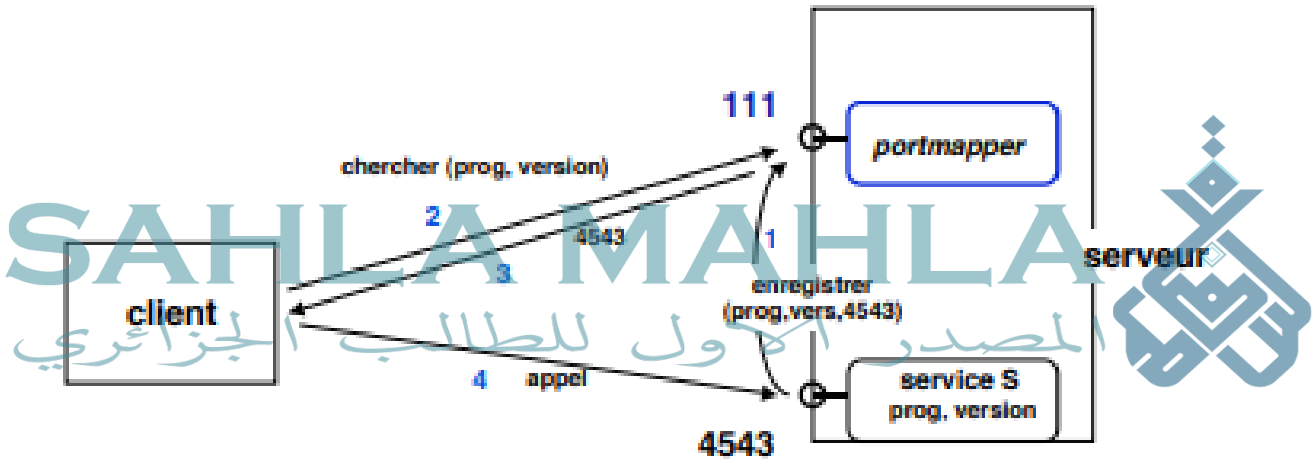


- 1, 2 : le serveur s'enregistre auprès de l'annuaire avec <nom, adr. serveur, n° port>
- 3, 4, 5 : le client consulte l'annuaire pour trouver <adr. serveur, n° port> à partir de <nom>

⇒ Désign. et liaison utilisant le portmapper

– Si le serveur est connu (cas fréquent) : on peut utiliser un service de nommage local au serveur, le `portmapper`.

– Un service enregistre le n° de port de son veilleur auprès du portmapper et le veilleur se met en attente sur ce port.
– Le portmapper a un n° de port fixé par convention (111).



d. Développement

Il existe trois façon de développer des programmes utilisant des RPC, utiliser :

⇒ Les fonctions de la **couche intermédiaire**

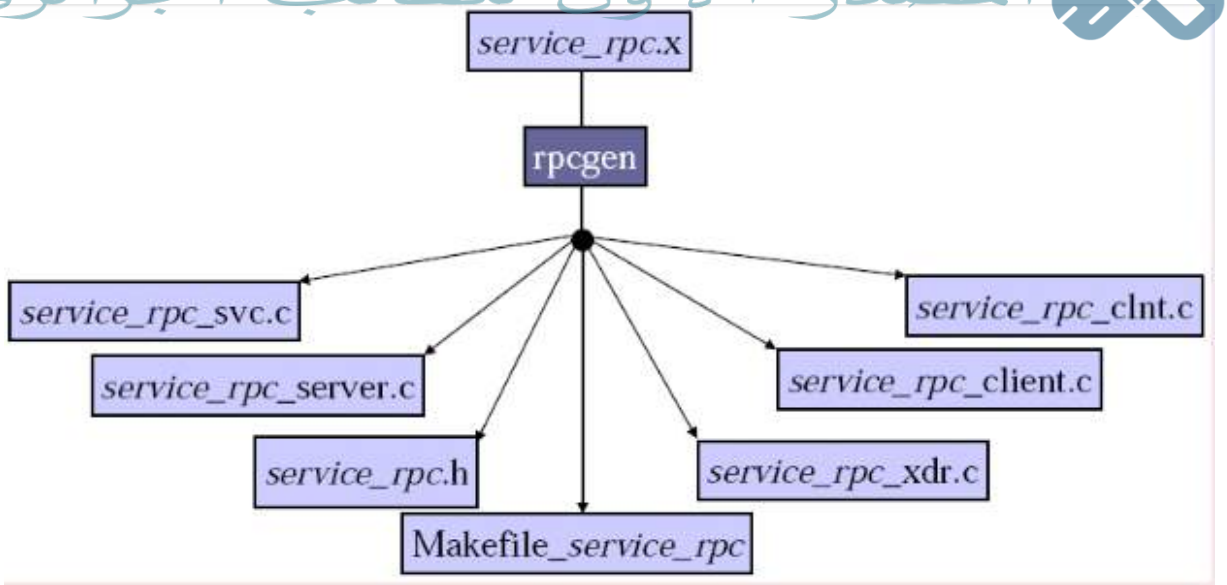
- Elle comporte peu de fonctions.
- Le programmeur est limité dans la configuration du système.

- le programmeur doit développer l'encodage et le décodage.

⇒ Les fonctions de la **couche basse**

- C'est un ensemble complet de fonctions.
- Son utilisation est beaucoup plus complexe que la couche intermédiaire.
- À n'utiliser que lorsque le protocole de communication et les délais de temporisation de la couche intermédiaire ne sont pas satisfaisant.

⇒ Le compilateur **rpcgen**



~ ∘ ◯ ∘ ~

4. CORBA (Common Object Request Broker Architecture)

a. .....

5. RMI (Remote Method Invocation)

- a. .....

6. JAVA EE

○○○

a. .....

Références