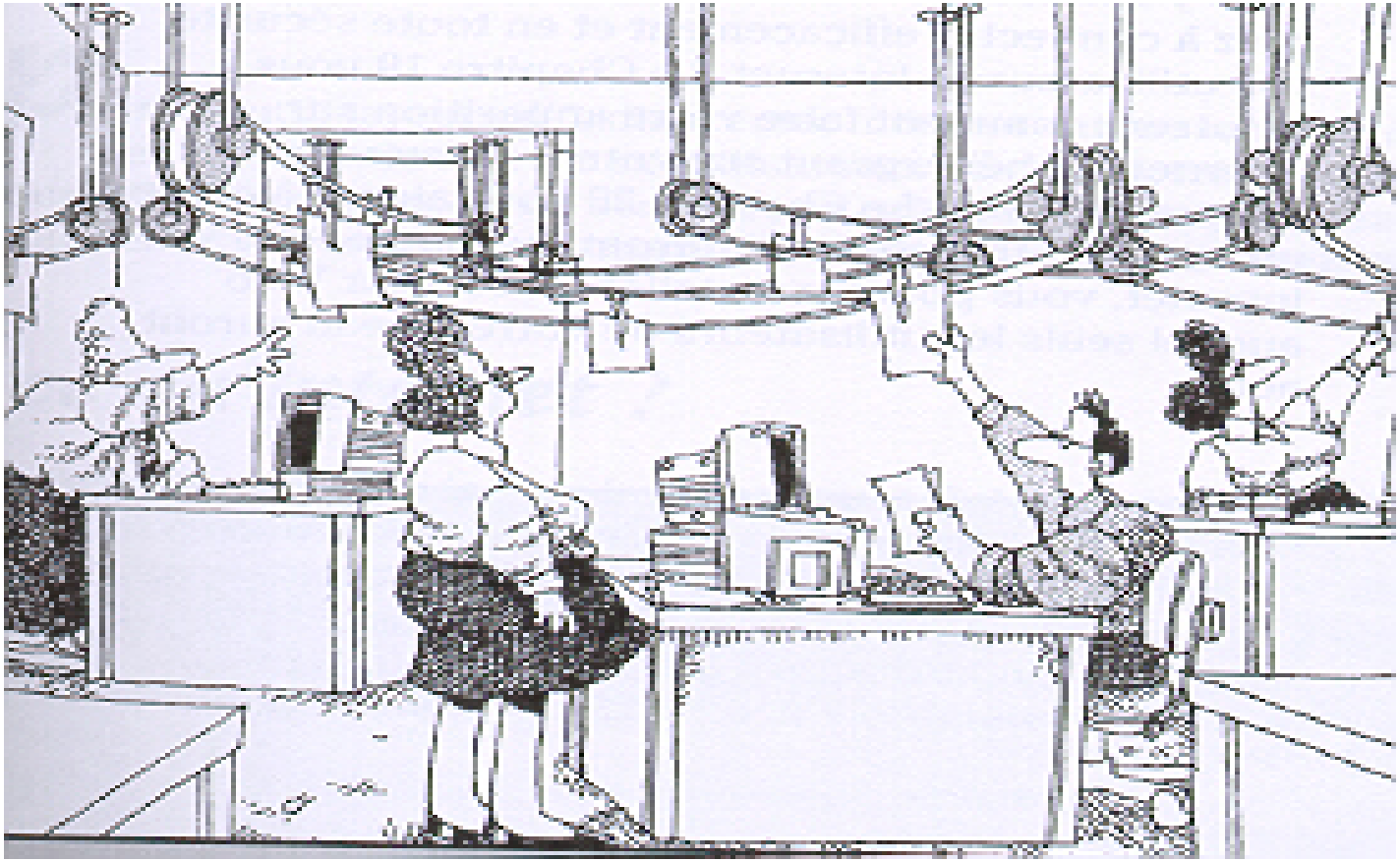
Introduction à la Programmation Réseau
Avec L'API SOCKET



Pr. A. HASBI

Les sockets

- IPC : Fichier, Passage de messages(files d'attente), Pipe (tube nommé, tube anonyme), Sémaphore, Mémoire partagée, Signaux, Sockets (INET et UNIX).

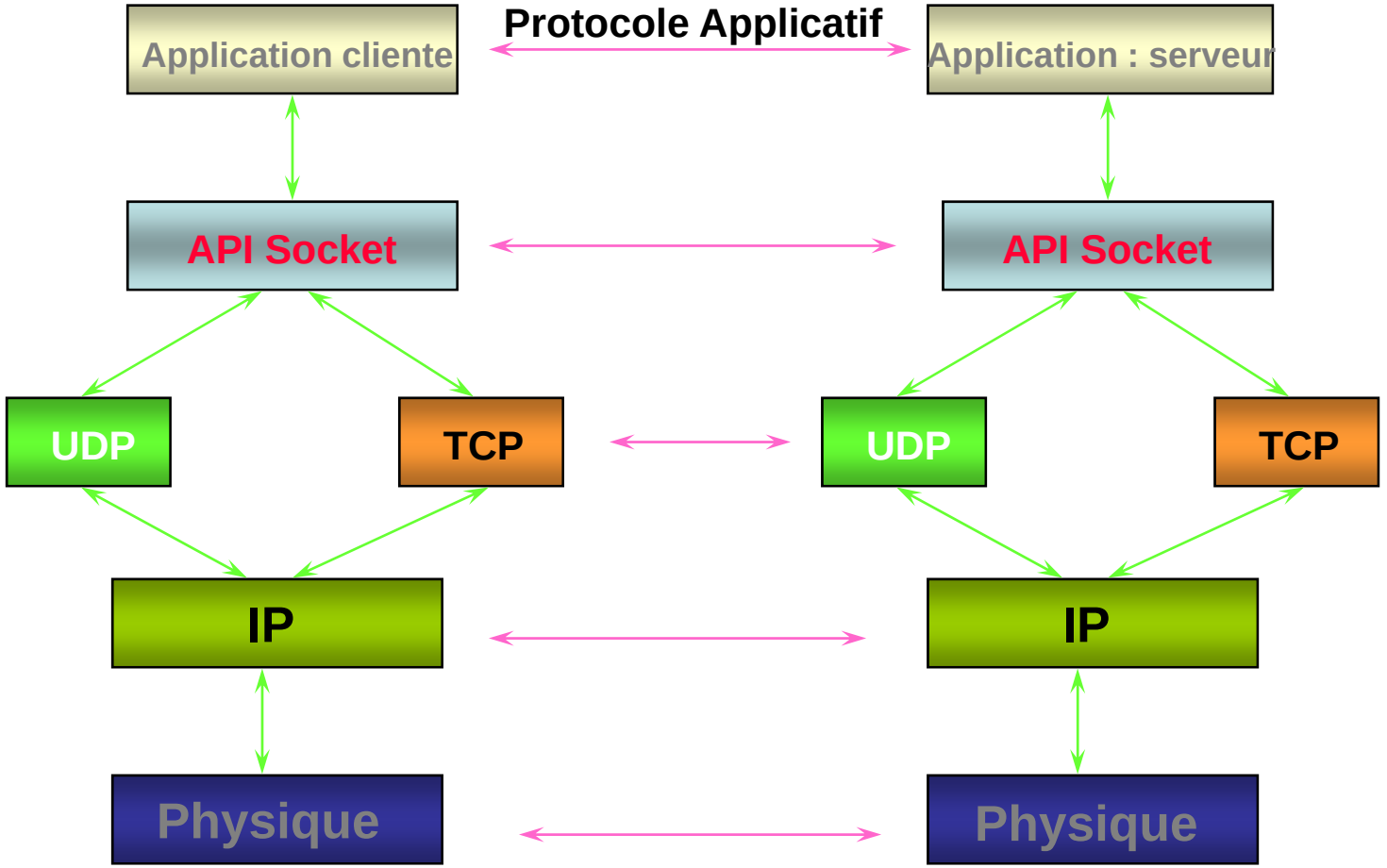
- Les sockets : interface client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP.

- Etendue aujourd'hui du micro (Winsock) au Mainframe.

- fournit les primitives pour le support des communications reposant sur toute suite de protocoles; les protocoles TCP/IP sont à l'origine des développements.

- Les applications client/serveur ne voient les couches de communication qu'à travers l'API socket (abstraction).

Les sockets


Application cliente
Protocole Applicatif
Application : serveur
API Socket
API Socket
UDP
TCP
UDP
TCP
IP
IP
Physique
Physique

Sockets : l'abstraction

- comme un descripteur de fichier dans le système UNIX,
- associe un descripteur à une socket;
- le concepteur d'application utilise ce descripteur pour référencer la communication client/serveur sous-jacente.
- une structure de données «socket» est créée à l' ouverture de socket;

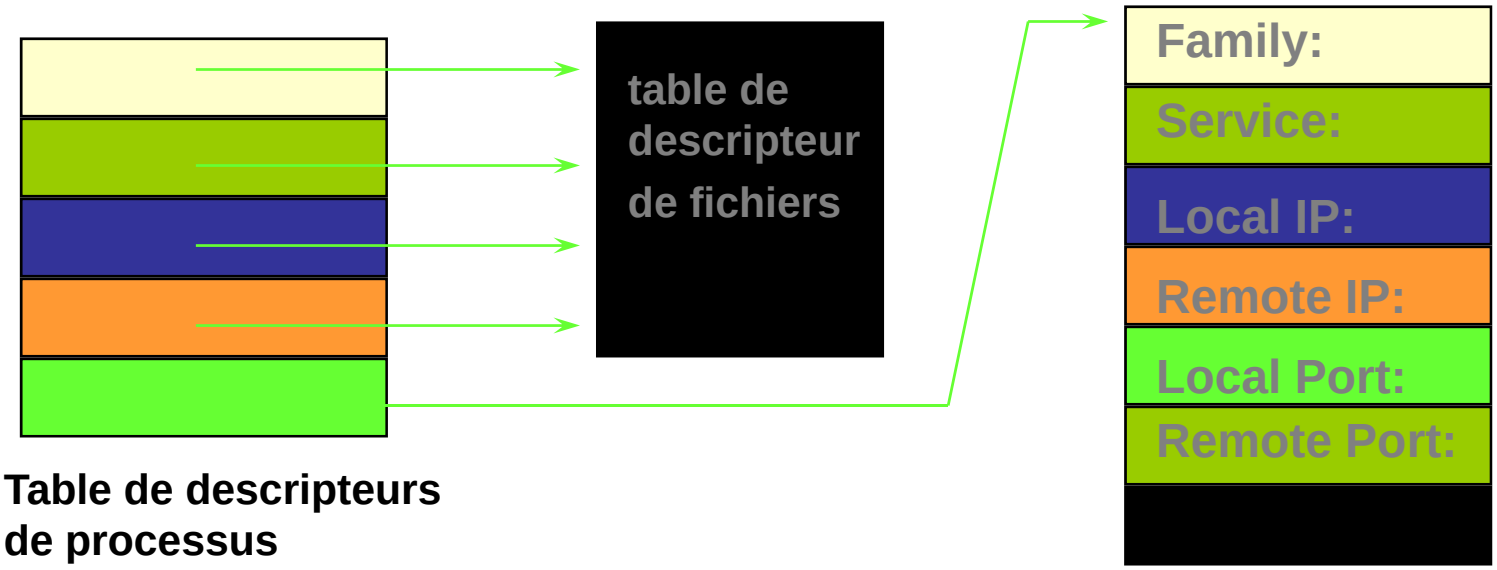


Table de descripteurs de processus

Structure Socket

La primitive *socket* permet l'ouverture de cette socket; initialement, après l'appel à cette fonction, la structure de données associée à une socket est principalement vide, les appels à d'autres primitives de l'interface socket renseigneront ces champs vides.

Les Sockets : primitives

- Elles permettent d'établir un lien de communication en mode connecté ou non-connecté sur un réseau,
- Structurent une application
  - soit en mode client ,
  - soit en mode serveur,
- Permettent d'échanger des données entre ces applications.
- La primitive *socket* : *socket(AF_INET,SOCK_STREAM,0)*
  - point d'ancrage qui permet à l'application d'obtenir un lien de communication vers la suite de protocoles qui servira d'échange,
  - définit le mode de communication utilisé (connecté ou non-connecté).
- La primitive *bind* : bind(s,(struct sockaddr*)&serveur,sizeof(serveur))

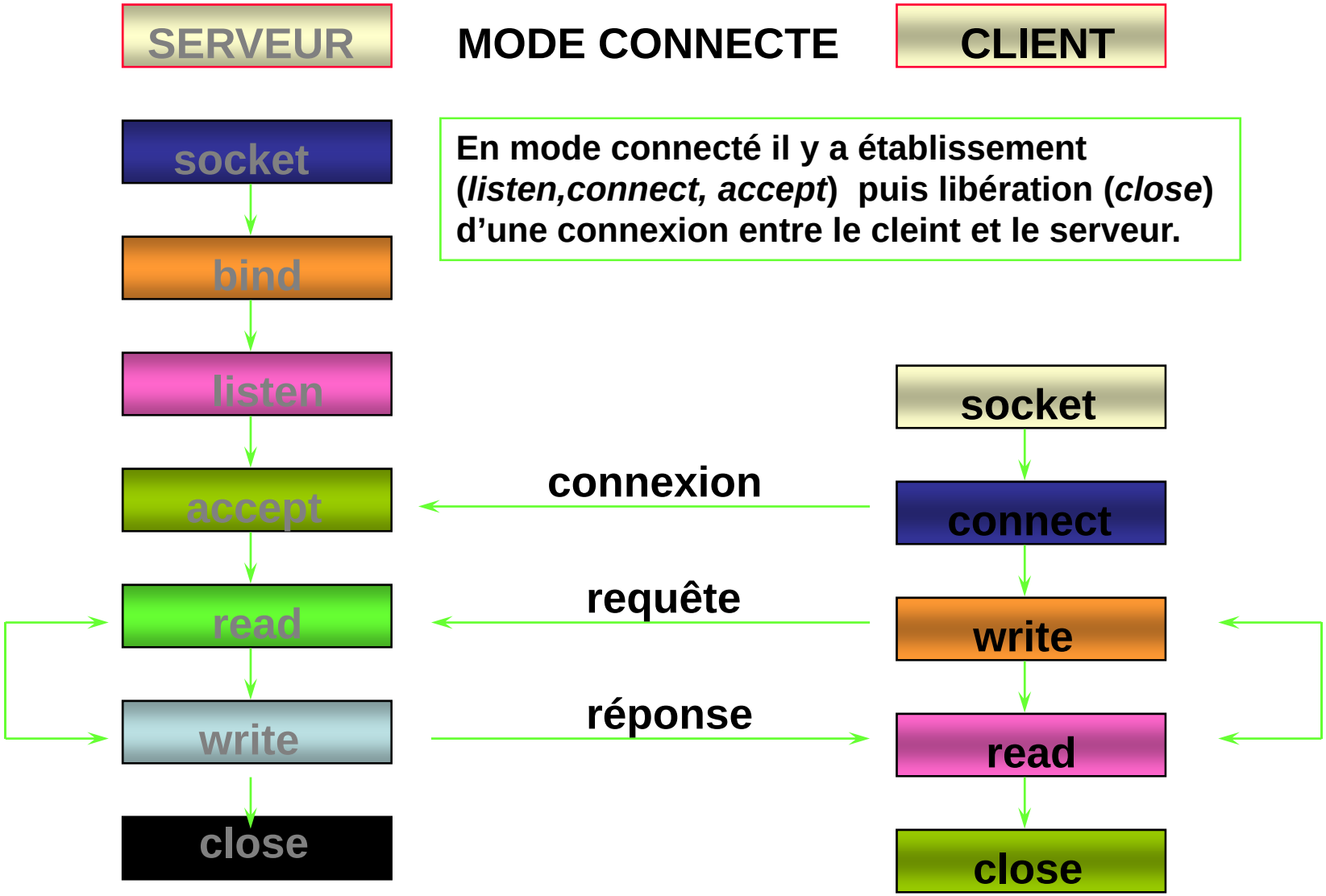
  permet de spécifier le point de terminaison local (essentiellement le port TCP/UDP dans l'environnement TCP/IP).
- la primitive *connect* : *connect(s,(struct sockaddr *)&serveur,sizeof(serveur))*
  - permet à un client d'établir une communication active avec un serveur,
  - le point de terminaison distant (adresse IP + port TCP/UDP dans l'environnement TCP/IP) est spécifié lors de cet appel.

Les Sockets : primitives

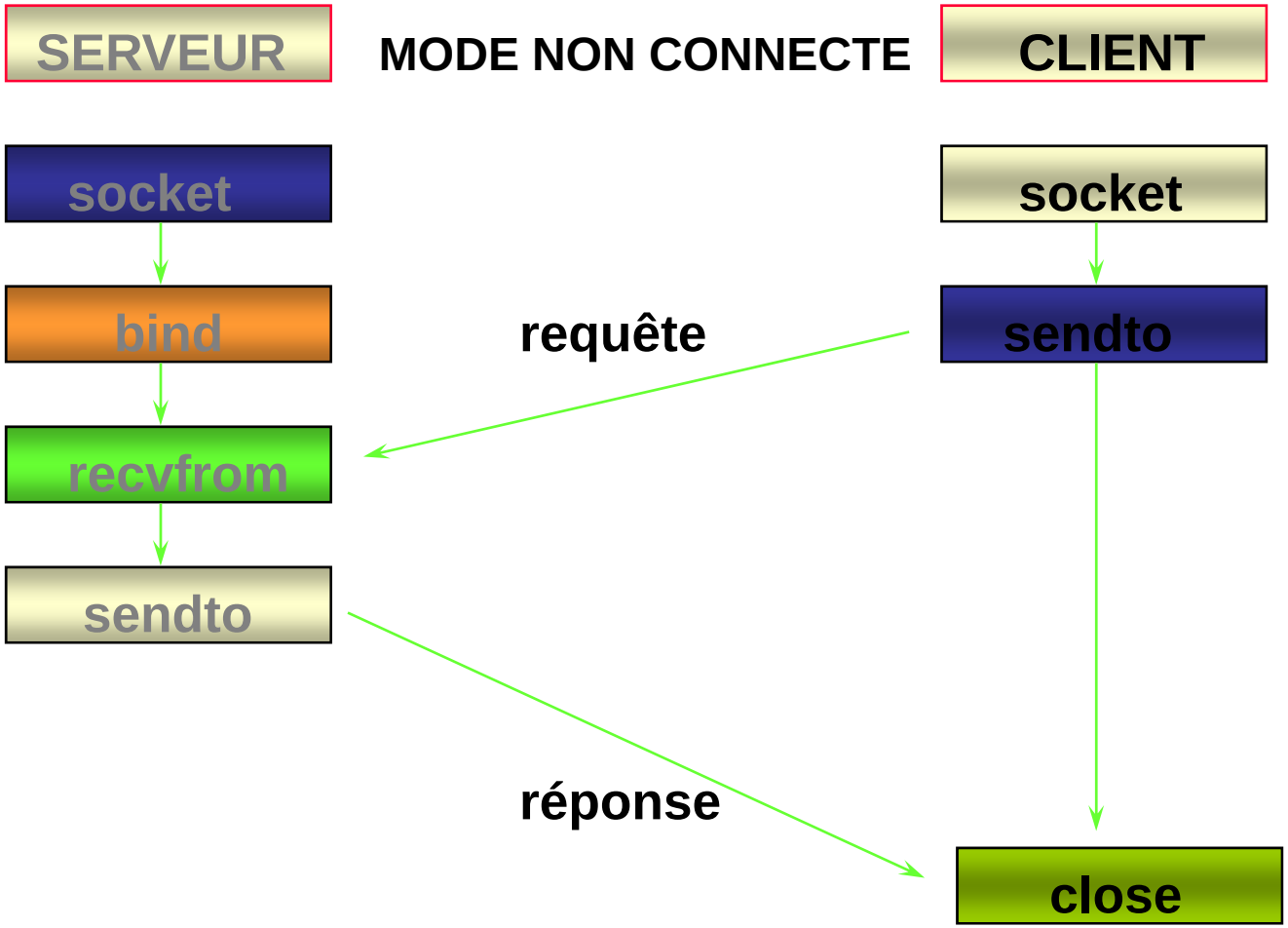
- la primitive *listen* : listen(s,5)
  - permet à un serveur d'entrer dans un mode d'écoute de communication ,
  - dés lors le serveur est « connectable » par un client,
  - le processus est bloqué jusqu'à l'arrivée d'une communication entrante.
- la primitive *accept* : newsc=accept(s,(struct sockaddr *)&client,&longueur))
  - permet à un serveur de recevoir la communication entrante (client),
  - crée une nouvelle socket et retourne le descripteur associé à l'application.
  - le serveur utilise ce descripteur pour gérer la communication entrante
  - le serveur utilise le descripteur de socket précédent pour traiter la prochaine communication à venir.
- les primitives *read* et *write:*
  - Lorsque la communication est établie, client et serveur échangent des données afin d'obtenir (client) et transmettre (serveur) le service désiré.
  - En mode connecté, clients et serveurs utilisent *read* (recv) et *write (*send*)*; en mode non-connecté, ils utilisent les primitives *recvfrom* et *sendto*.
- la primitive *close* :

  termine la connexion et libère le socket associé.

Les Sockets : Mode connecté



Les Sockets : Mode non connecté



Socket : Mode non connecté

En mode non-connecté:

- le client n'établit pas de connexion avec le serveur mais émet un datagramme (sendto) vers le serveur.
- Le serveur n'accepte pas de connexion, mais attend un datagramme d'un client par recvfrom qui transmet le datagramme à l'application ainsi que l'adresse client.

- Les sockets en mode non-connecté peuvent utiliser la primitive connect pour associer une socket à une destination précise. ==> send peut être utilisée à la place de sendto.
- De même, si l'adresse de l'émetteur d'un datagramme n'intéresse pas un processus la primitive recv peut être utilisée à la place de la primitive recvfrom.

Socket : exemple de serveur itératif

```
int sockfd, newsockfd ;

if ( ( sockfd = socket (.....)) < 0 )  err_sys(«erreur de socket«) ;
if ( bind ( sockfd, ....) < 0 )                    err_sys («erreur de bind»)
if ( listen ( sockfd , 5) < 0 ) ;                  err_sys (« erreur de listen» ) ;

for ( ; ; )  {
    newsockfd = accept ( sockfd, .....) ;
    if ( newsockfd < 0)
            err_sys( «erreur de accept») ;

    execute_la_demande( newsockfd ) ;
    close ( newsockfd ) ;
}
```

Socket : exemple de serveur parallèle

```
int sockfd, newsockfd ;

if ( ( sockfd = socket (.....)) < 0 )       err_sys(«erreur de socket«) ;
if ( bind ( sockfd, ....) < 0 )  err_sys («erreur de bind»)
if ( listen ( sockfd , 5) < 0 ) ;           err_sys (« erreur de listen» ) ;

for ( ; ; )  {
    newsockfd = accept ( sockfd, .....) ;
    if ( newsockfd < 0)        err_sys( «erreur de accept») ;
    if ( fork() == 0 ) {
            close ( sockfd ) ;
            execute_la_demande( newsockfd ) ;
            exit (1) ;
            }
    close ( newsockfd ) ;
}
```

Sockets : gestion de noms

- Lorsque ces fonctions sont exécutées sur des machines ayant accès à un serveur de noms de domaines, elles fonctionnent elles-mêmes en mode client/serveur en émettant une requête vers le serveur de nom de domaines et attendent la réponse.
- Lorsqu'elles sont utilisées sur des machines qui n'ont pas accès à un serveur de noms, elles obtiennent les informations à partir d'une base de données (simple fichier) locale.

- *gethostbyname* spécifie un nom de domaine et retourne un pointeur vers une structure *hostent* qui contient les informations propres à ce nom de domaine.
- *gethostbyaddr* permet d'obtenir les mêmes informations à partir de l'adresse spécifiée.
- *getnetbyname* spécifie un nom de réseau et retourne une structure *netent* renseignant les caractéristiques du réseau.
- *getnetbyaddr* spécifie une adresse réseau et renseigne la structure *netent*

Sockets : fonctions de service

- Les fonctions getprotobyname et getprotobynumber
  - Dans la base de données des protocoles disponibles sur la machine, chaque protocole a un nom officiel, des alias officiels et un numéro de protocole officiel.
  - La fonction *getprotobyname* permet d'obtenir des informations sur un protocole donné en spécifiant son nom; renseigne la structure *protoent*.
  - La fonction *getprotobynumber* permet d'obtenir les mêmes informations en spécifiant le numéro de protocole.

- La fonction getservbyname
  - Certains numéros de ports sont réservés pour les services s'exécutant au-dessus des protocoles TCP et UDP.
  - *getservbyname* retourne les informations relatives à un service donné en spécifiant le numéro du port et le protocole utilisé; renseigne la structure *servent*.

Sockets : Byte ordering

- TCP/IP spécifie une représentation normalisée pour les entiers utilisés dans les protocoles. Cette représentation, appelée *network byte order*, représente les entiers avec le MSB en premier.
- Une application doit renseigner certaines informations du protocole et par conséquent, doit respecter le *network* informations *byte order;* Exemple le numéro de port.
- Pour que les applications fonctionnent correctement, elles doivent translater la représentation des données de la machine locale vers le *network byte order* :
  - *htonl* : host to network long : convertit une valeur sur 32 bits de la représentation machine vers la représentation réseau.
  - *htons* : host to network short : convertit une valeur sur 16 bits de la représentation machine vers la représentation réseau.
  - *ntohl* : network to host long : convertit une valeur sur 32 bits de la représentation réseau vers la représentation machine.
  - *ntohs* : network to host short : convertit une valeur sur 16 bits de la représentation réseau vers la représentation machine.

Sockets : gestion de noms

- Les primitives gethostname et sethostname
  - Dans le monde UNIX, la primitive *gethostname* permet aux processus utilisateurs d'accéder au nom de la machine locale.
  - D'autre part, la primitive *sethostname* permet à des processus privilégiés de définir le nom de la machine locale.
- La primitive getpeername
  - Cette primitive est utilisée afin de connaître le point de terminaison du distant.
  - Habituellement, un client connaît le point de terminaison (couple port/adresse IP) puisqu'il se connecte à ce serveur distant; cependant, un serveur qui utilise la primitive *accept* pour obtenir une connexion, a la possibilité d'interroger la socket afin de déterminer l'adresse du distant.
- La primitive getsockname
  - Cette primitive rend le nom associé à la socket qui est spécifiée en paramètre.

Bibliothèques & Structures

- struct sockaddr {
  unsigned short sa_family; /* famille d'adresse, AF_xxx */
  char sa_data[14]; /* 14 octets d'adresse de protocole */
  };
- struct sockaddr_in {
  short int sin_family; /* Famille d'adresse */
  unsigned short int sin_port; /* Numéro de Port */
  struct in_addr sin_addr; /* Adresse Internet */
  unsigned char sin_zero[8]; /* Même taille que struct sockaddr */
  };
- /* Internet adresse (une structure pour des raisons historique) */
- struct in_addr {
  unsigned long s_addr;
  };
- #include <netdb.h>
- struct hostent *gethostbyname(const char *name); struct hostent { char *h_name; char **h_aliases; int h_addrtype; int h_length; char **h_addr_list; };

Bibliothèques & Structures

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

```
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
};
```

- #include <sys/types.h><netinet/in.h><sys/socket.h>

Sockets : les options

- Une application peut contrôler certains aspects du fonctionnement des sockets:

  – configurer les valeurs des temporisations,
  – l'allocation de la mémoire tampon,
  – vérifier si la socket autorise la diffusion ou la gestion des données hors bande.
- La primitive getsockopt

- Permet à une application d'obtenir les informations relatives au socket. Le système d'exploitation exploite les structures de données internes relatives au socket et renseigne l'application appelante.

Sockets : les options

level	optname	get	set	Description	flag	type de données
PPROTO_IP	IP_OPTIONS	•	•	option de l'entête IP		
PPROTO_TCP	TCP_MAXSEG	•		donne la taille max d'un segmen	•	int
	TCP_NODELAY	•	•	ne pas retarder l'envoi pour grouper des paquets		int
OL_SOCKET	SO_DEBUG	•	•	permet des infos de debugging	•	int
	SO_DONTROUTE	•	•	utilise uniquement les adresses d'interface	•	int
	SO_ERROR	•		rend le status de l'erreur	•	int
	SO_LINGER	•	•	contrôle de l'envoi des données après close		struct linger
	SO_OOBINLINE	•	•	concerne la réception de donnée hors bande	•	int
	SO_RCVBUF	•	•	taille du buffer de réception		int
	SO_SNDBUF	•	•	taille du buffer d'envoi		int
	SO_RCVTIMEO	•	•	timeout de réception		int
	SO_SNDTIMEO	•	•	timeout d'emission		int
	SO_REUSEADDR	•	•	autorise la réutilisabilité de l'adresse locale	•	int
	SO_TYPE	•		fournit le type de socket		int

Sockets : serveur multi-protocoles

- Certains serveurs offrent leurs services sur plusieurs protocoles simultanément afin de satisfaire les clients qui nécessitent des transports, soit en mode connecté, soit en mode non-connecté.
  - Exemple : DAYTIME port 13 sur UDP et sur TCP.
- Les services réalisés sur l'une ou l'autre interface fonctionnent différemment :
  - la version TCP utilise la connexion entrante du client pour déclencher la réponse (à une requête donc implicite): le client n'émet aucune requête.
  - la version UDP de DAYTIME requiert une requête du client. Cette requête consiste en un datagramme arbitraire nécessité pour déclencher l'émission de la donnée côté serveur. Ce datagramme est ensuite rejeté par le serveur.
- Dans de nombreux cas, un serveur fournit un service pour un protocole donné; par exemple, le service DAYTIME est réalisé par deux serveurs différents, l'un servant les requêtes TCP, l'autre les requêtes UDP.

Serveurs multi-protocoles

- Avantage à utiliser des serveurs différents réside dans le contrôle des protocoles et des services qu'offrent un système (certains systèmes, par exemple, ferment tout accès à UDP pour des raisons de sécurité).

- L'avantage à utiliser un serveur commun ou multi-protocoles :
  - non duplication des ressources associées au service, (corps du serveur),
  - cohérence dans la gestion des versions.

- Fonctionnement

  - Un seul processus utilisant des opérations d'entrée/sortie asynchrones de manière à gérer les communications à la fois en mode connecté et en mode non-connecté.
  - Deux implémentations possibles : en mode itératif et en mode concurrent.

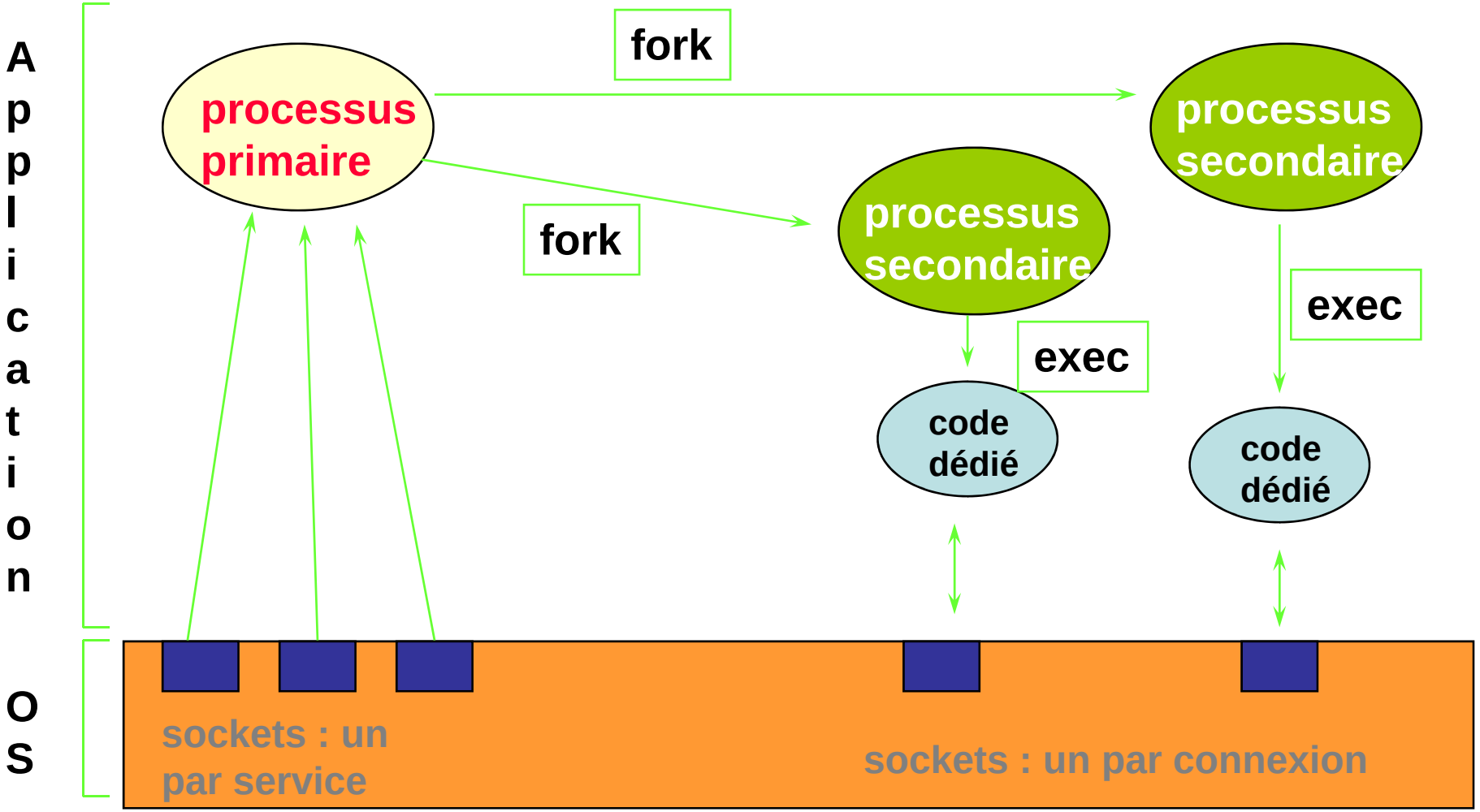
Serveurs multi-protocoles

- En mode itératif
  - le serveur ouvre un socket UDP et un socket TCP,
  - Lorsqu'une requête TCP arrive, le serveur utilise *accept* provoquant la création d'un nouveau socket servant la communication avec le client,
  - Lorsque la communication avec le client est terminée, le serveur ferme le troisième socket et réitère son attente sur les deux sockets initiales.
  - Si une requête UDP arrive, le serveur reçoit et émet des messages avec le client (il n'y a pas d'*accept*); lorsque les échanges sont terminés, le serveur réitère son attente sur les deux sockets initiales
- Le mode concurrent
  - Création d' un nouveau processus pour toute nouvelle connexion TCP et traitement de manière itérative des requêtes UDP.
  - Automate gérant les événement asynchrones :optimisation maximale des ressources machines, puisque un seul processus traite toutes les variantes protocolaires des serveurs (TCP et UDP) et toutes les instances de services seront également traitées par le même processus.

Sockets : les serveurs multi-services

- Problème lié à la multiplication des serveurs : le nombre de processus nécessaires et les ressources consommées qui sont associées.
- La consolidation de plusieurs services en un seul serveur améliore le fonctionnement:
- La forme la plus rationnelle de serveur multi-services consiste à déclencher des programmes différents selon la requête entrante : le fonctionnement d'un tel serveur en mode connecté est le suivant:
  - le serveur ouvre un socket par service offert,
  - le serveur attend une connexion entrante sur l'ensemble des sockets ouverts,
  - lorsqu'une connexion arrive, le serveur crée un processus secondaire (*fork* sous système UNIX), qui prend en compte la connexion,
  - le processus secondaire exécute (via *exec* sous système UNIX) un programme dédié réalisant le service demandé.

Sockets : serveurs multi-services



Sockets : Serveurs multi-services

AVANTAGES

- le code réalisant les services n'est présent que lorsqu'il est nécessaire,

- la maintenance se fait sur la base du service et non du serveur : l'administrateur peut gérer le serveur (modifie, archiver, ...) par service au lieu de le gérer globalement.

- Ce schéma est retenu en standard : le « super serveur » (*inetd* en BSD) consistant en un processus multi-services multi-protocoles offrant une interface de configuration (fichier systèmes) permettant à l'administrateur système d'ajouter de nouveaux services alors qu'aucun processus supplémentaire n'est nécessaire.