

Les Sockets

cb

/home/kouna/d01/adp/bcousin/Fute/Cours/Internet/06-socket.fm- 12 Octobre 1998 17:27)

Plan

- Présentation
- L'enchaînement des primitives
- Les primitives
- Conclusion
- Quelques informations supplémentaires

Bibliographie

- L.Toutain, Réseaux locaux et Internet, Hermès, 1996.
- M.Gabassi, B.Dupouy, L'informatique répartie sous Unix, Eyrolles, 1992.
- J-M.Rifflet, La communication sous Unix, EdiScience international, 1995.
- D.E.Comer, Internetworking with TCP/IP, Prentice-Hall, 1991.

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

1. Présentation

1.1. Définition

Les Sockets : Interface de programmation pour les communications

- ❑ Ensemble de primitives assurant ce service,
- ❑ Générique : s'adapte aux différents besoins de communication,
- ❑ Indépendant de protocoles et de réseaux particuliers :
 - . Mais développé à l'origine sous Unix 4BSD, pour Internet !
- ❑ N'utilise pas forcément un réseau :
 - . Par exemple : communication locale (interne à une station) : domaine Unix !

Une Socket : un point de communication par lequel un processus peut émettre ou recevoir des données

- ❑ Homogène avec les identificateurs d'E/S :
 - . l'identificateur (de descripteur) de Socket est compatible avec l'identificateur (de descripteur) de fichiers.
- ❑ On distingue la création de la Socket : `socket()`, de son initialisation avec les adresses et les numéros de port : `bind()`, `connect()`, à contrario des fichiers : `open()` !

1.2. Les domaines

Les sockets peuvent gérer plusieurs familles protocolaires :

- Internet : PF_INET
- fichiers locaux : PF_UNIX
- OSI, SNA, DEC, CCITT(X25), Appletalk,
- etc.

1.3. Serveur/Client

Le paradigme du Client/Serveur est extrêmement courant au sein des applications réparties

Un serveur :

- processus rendant un service spécifique identifié par un port particulier (n° port),
- 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.

1.4. Les structures de données

1.4.1 La structure `socket`

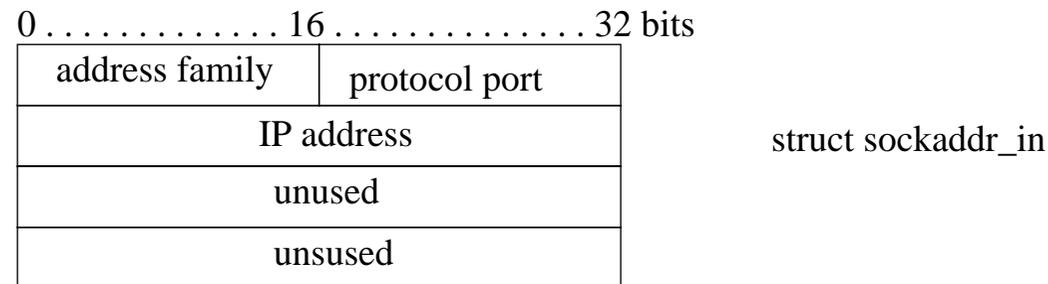
Descripteur du point d'accès à la communication :

- type, options, état, temporisateurs, liste des processus en attente, tampons d'émission et de réception, etc.

Structure décrite dans le fichier `<sys/socketvar.h>`

1.4.2 La structure `sockaddr`

Adresse : désignation spécifique permettant d'être identifié de l'extérieur



Structure décrite dans le fichier `<netinet/in.h>`

Structure polymorphe (générique) :

- adaption à la famille protocolaire utilisée :
 - . par exemple : sockaddr_un, sockaddr_in.

1.4.3 La structure `hostent`

Chaque système maintient localement (ou accède à) une base de données contenant des informations sur les stations du réseau.

Structure définie dans le fichier `<netdb.h>` :

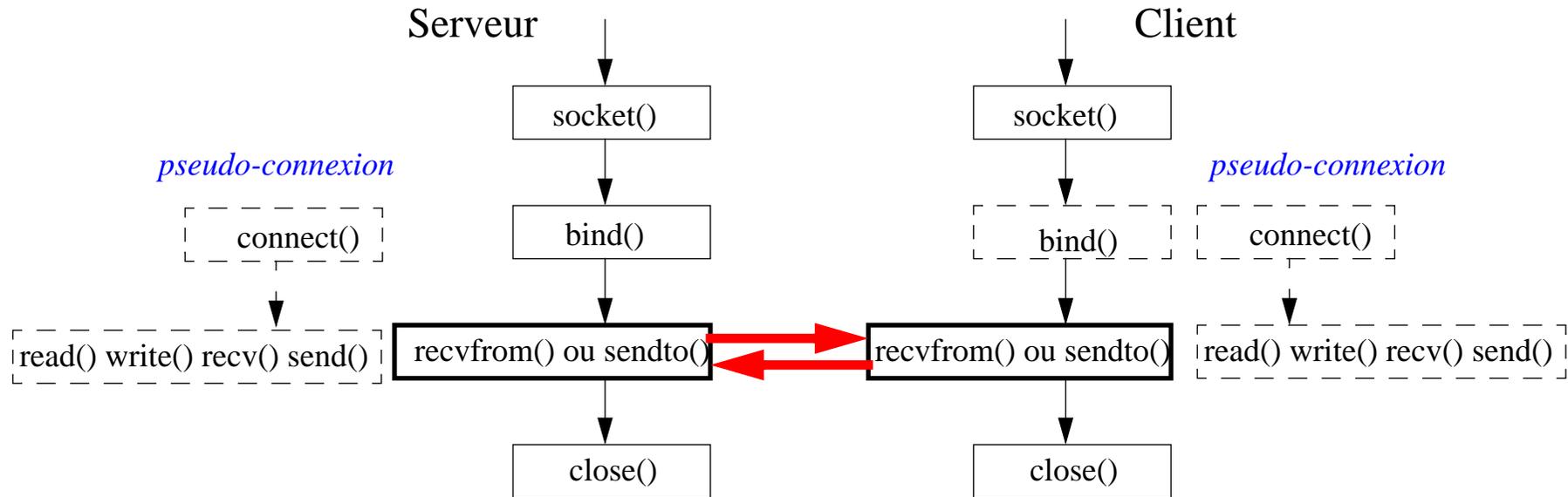
- liste de noms,
- liste d'adresses,
- type d'adresse et longueur;

Informations :

- issues des fichiers `/etc/hosts`, `/etc/config`, `/etc/nsswitch.conf`, etc.
- ou obtenues à l'aide de DNS ou NIS.

2. L'enchaînement des primitives

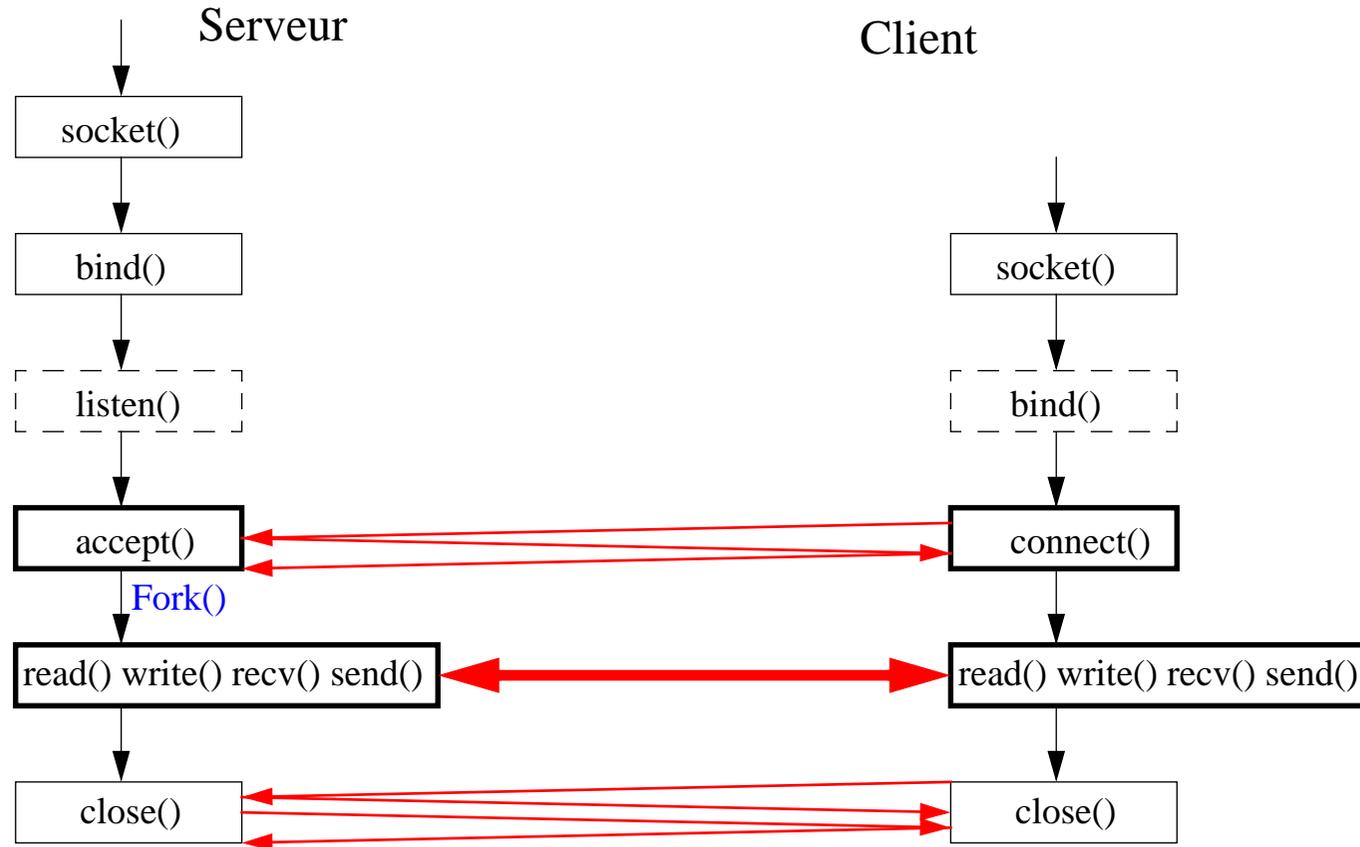
2.1. Exemple d'enchaînement simple en mode non-connecté



généralement bloquant
 optionnel
 ➔ transfert de données

➔ **UDP**

2.2. Exemple d'enchaînement simple en mode connecté



Chaque "close()" ne ferme qu'un seul sens de communication !

➔ TCP

3. Les primitives

3.1. La création d'une Socket

int **socket**(int *domain*, int *type*, int *protocol*);

descripteur
de la socket,
-1 si erreur

domaine de
communication :
PF_UNIX ou
PF_INET, etc.

voir liste dans :
<sys/socket.h>

mode de
communication :
SOCK_STREAM,
SOCK_DGRAM,
etc.

voir liste dans :
<sys/socket.h>

protocole à
utiliser,
(0 => le
système
choisit le
protocole
le mieux
adapté au
mode).

voir dans :
<netinet/in.h>

Exemple : s=socket(PF_INET,SOCK_DGRAM,0);

3.2. Renseignement de la Socket avec les informations locales

```
int bind(int s, struct sockaddr_in * addsock, int lg_addsock);
```

0 si succès
-1 si erreur

id. local
de la
socket
créée

pointeur sur la structure
contenant les informations
sur la machine locale
(domaine, n° de port,
et adresse IP)

cf <netinet/in.h>

taille de la
structure
précédente
(obtenue
avec sizeof)

INADDR_ANY: remplace n'importe quelle adresse

si le numéro de port n'est pas précisé, il est choisi par le noyau

Exemple :

```
if (bind(s, &add_sock, sizeof(add_sock)))
    {fprintf(stderr, "Echec dans la réalisation du bind(...)");
    exit(EXIT_BINDING_SOCKET);
    }
```

3.3. Longueur maximum de la file d'attente des communications en cours d'établissement

int *listen*(int *s*, int *backlog*);

0 si succès
-1 si erreur

id. de la
socket
locale

taille de la file d'attente des
demandes de communication
(par défaut= 5)

Exemple :

```
if (listen(sock,MAXCONREQ) < 0)
{perror("Echec du listen");
  exit(EXIT_ERR_LISTEN);
}
```

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

3.4. Définition de l'ensemble des connexions acceptables

```
int accept(int s, struct sockaddr_in * from, int * fromlen);
```

↓
id. de la **nouvelle** socket
ou -1 si erreur

← la structure contenant
les informations sur
les émetteurs acceptables

← la taille de la
structure est
mise ici

```
Exemple : new_sock = accept(s, &add_initiator, &size_add_initiator);
            if (new_sock < 0)
                { perror("Echec de l'accept");
                  exit(EXIT_ERR_ACCEPT);
                }
```

3.5. Renseignement de la Socket pour le destinataire

Etablissement de la (pseudo-)connexion

```
int connect(int s, struct sockaddr_in * dest, int lg_dest);
```

0 si succès
-1 si erreur

id. de la
socket
locale

pointeur sur la structure
contenant les informations
sur la **socket distante**
(récupérable avec
gethostbyname(...))

taille de la
structure
précédente

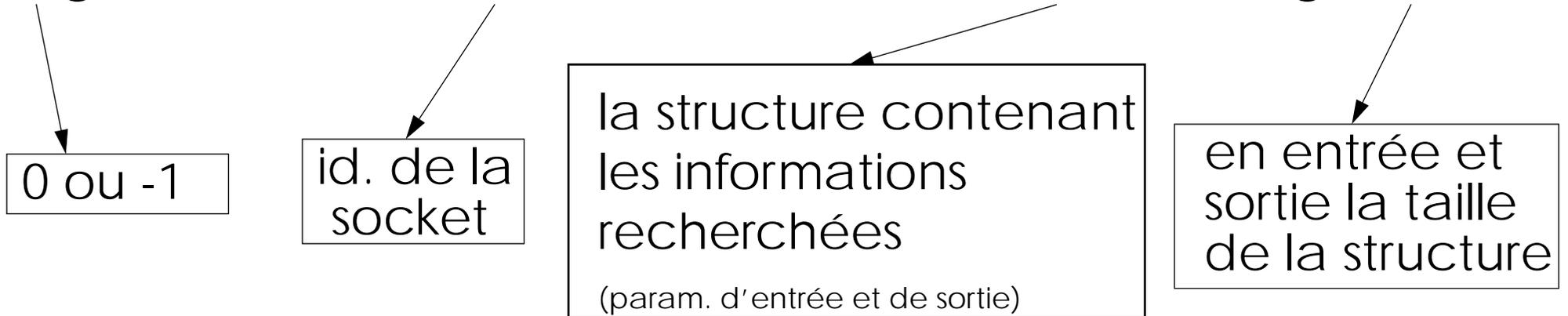
Exemple :

```
if (connect(sock, &addsock, sizeof(addsock)) < 0)
    {perror("Echec de lien de socket");
    exit(EXIT_ERR_CONNECT_SOCKET);
    }
```

3.6. Primitives d'accès aux informations sur la Socket

Obtenir l'adresse IP d'une socket dont on connaît le descripteur

```
int getsockname(int s, struct sockaddr_in * sk_inf, int * lg_sk_inf);
```



```
int getpeername(int s, struct sockaddr_in * sk_inf, int * lg_sk_inf);
```

joue le même rôle que `getsockname()` mais pour accéder à la socket distante avec laquelle on a établi au préalable une association (utilisable dans les 2 modes)

3.7. Primitives de réception de données

Strictement identiques aux primitives habituelles d'E/S :

```
int read(int s, char * buf, int nbyte);
```

nombre d'octets lus
0 si la connexion est fermée
ou -1 si erreur

les données reçues
sont mises dans la
zone pointée par *buf*

nombre
maximum
d'octets
à lire

Exemple :

```
if ((nblu=read(new_sock, buf, MAXBUFFER)) <= 0)
    {perror("Echec en réception de données");
      exit(EXIT_ERR_RECV_STREAM);
    }
```

3.8. Autres primitives de réception de données

Similaire au “read()” :

```
int recvfrom(int s, char * buf, int nbyte, int flag,
             struct sockaddr_in * from, int * fromlen);
```

=> utile surtout en mode non-connecté pour identifier l'émetteur

```
int recvmsg(int s, struct msghdr * msg, int flag);
```

=> pour réduire le nombre de paramètres à fournir directement

```
int recv(int s, char * buf, int nbyte, int flag);
```

=> utilisable plutôt en mode connecté

3.9. Primitives d'émission de données

Identiques ou similaires au “write()” habituel :

```
int write(int s, char * buf, int nbyte);
```

```
int sendto(int s, char * buf, int nbyte, int flag,  
           struct sockaddr_in * dest, int * destlen);
```

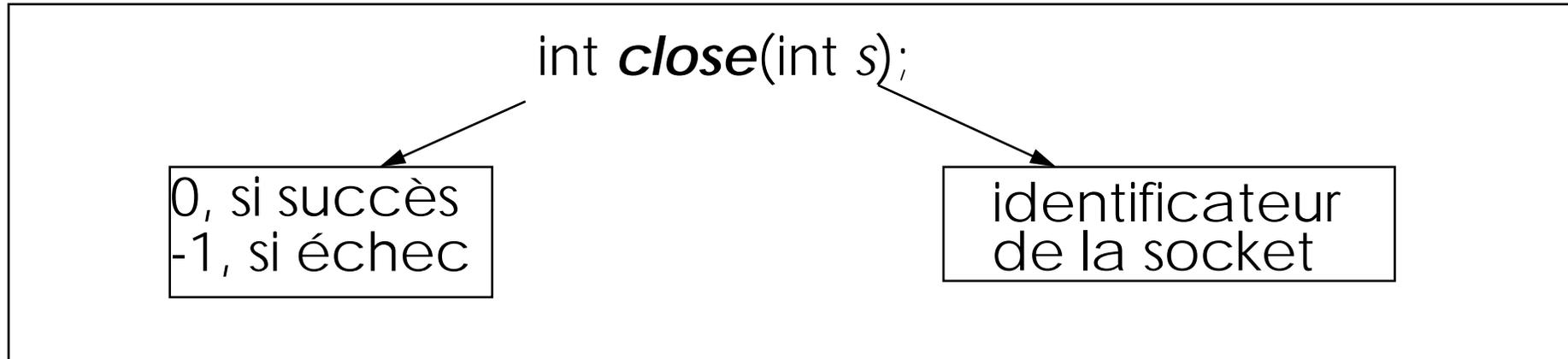
```
int sendmsg(int s, struct msghdr * msg, int flag);
```

```
int send(int s, char * buf, int nbyte, int flag);
```

3.10. Primitive de libération de la socket

Libère l'espace de stockage associé à la Socket,

Libère la connexion (si elle existe !)



3.11. Primitives d'interrogation de la base de données relatives aux sites

- pour obtenir le nom de la machine locale :

```
int gethostname(char * name, int namelen);
```

0 ou -1

le nom de machine
(tronqué à namelen car.)
est renvoyé dans name

- pour obtenir l'adresse d'une machine :

```
struct hostent * gethostbyname(char * name);
```

voir définition de la structure dans **<netdb.h>**

3.12. Autres primitives d'interrogation de la base de données :

- pour obtenir des **infos sur une machine** dont on connaît l'adresse :
 - . `gethostbyaddr()`
- pour connaître le **protocole** utilisé :
 - . `getprotobyname()`, `getprotobyname()...`
- pour identifier le **réseau** sur lequel on travaille :
 - . `getnetbyname()`, `getnetbyaddr()...`
- pour avoir des informations sur un **service** donné :
 - . `getservbyname()`, `getservbyport()...`

*(voir liste des services dans le fichier **/etc/services**)*

NB : Toutes les structures de données utilisées et/ou renvoyées par ces fonctions sont définies dans **<netdb.h>**

4. Conclusion

Il existe de nombreuses autres interfaces de programmation pour les communications. Par exemple :

- WinSock
 - . proposé pour IBM PC
- TLI (“transport level interface”)
 - . proposé par Unix System V
 - . implémentation par Stream
 - . compatible Socket

Pour d’autres informations :

- Le man !
- les documentations des constructeurs,
- les livres,
- les exemples d’applications existantes !!

5. Encore plus d'informations diverses

5.1. inetd : /etc/inetd.conf

/etc/services

5.2. Les signaux :

SIGIO : socket est prête pour une entrée-sortie

SIGURG : données urgentes sont disponibles dans une socket

SIGPIPE : il n'est plus possible d'écrire sur une socket

5.3. Les émissions asynchrones :

- select()
- read/write() non-bloquant

5.4. Les fonctions de contrôle

- ioctl() : paramètre FIONBIO, fcntl() : paramètre O_NDELAY
- setsockopt(), getsockopt()

5.5. Traduction de notation d'adresse entre "dotted decimal" et adresse IP :

- `address = inet_addr(string);`
- `string = inet_ntoa(inet_addr);`

5.6. Consultation de la base de données réparties : DNS

Recherche dans la BdD par nom ou par adresse :

- `info_station = gethostbyname(nom_station);`
- `info_station = gethostbyaddr(adresse_station);`
- `sethostent(), gethostent(), endhostent()`