

## Les Sockets

(Z:\Polys\Internet\_transmission\_donnees\06-socket.fm- 18 février 2009 16:52)

### 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.

Transparents basés sur ceux aimablement communiqués par César Viho.

## 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 de communication

Les sockets peuvent gérer plusieurs **domaines de communication** :

- Internet : PF\_INET
- fichiers locaux : PF\_UNIX
- OSI, SNA, DEC, CCITT(X25), Appletalk,
- etc.

## 1.3. Serveur/Client

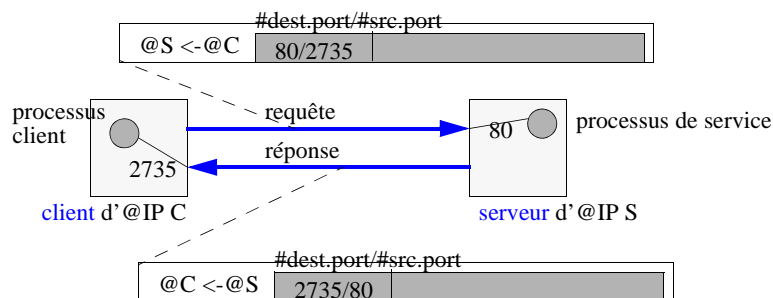
Le paradigme du Client/Serveur est extrêmement courant au sein des applications réparties.

Un serveur :

- processus rendant un **service** spécifique identifié par un **port** particulier bien connu (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 du réseau.



Abus de langage :

- on confond souvent la station supportant le processus et le processus serveur (resp. client). Cependant une même station peut supporter plusieurs serveurs et clients, et pour des services différents.

## 1.4. Les structures de données

3 structures de données :

- la socket,
- l'adresse,
- l'entrée dans la base de données des stations

### 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.

Rôle équivalent au descripteur d'E/S : inode/vnode.

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

Manipulée par les primitives : `socket()`, `bind()`, `accept()`, `connect()`, `read()`, `write()`, `close()`, etc.

### Extrait du fichier `<sys/socketvar.h>`

*/\* The sonode represents a socket. A sonode never exist in the file system name space and can not be opened*

*\* using open() - only the socket, socketpair and accept calls create sonodes \*/*

```
struct sonode {
    struct vnode so_vnode;           /* vnode associated with this sonode */
    dev_t so_dev;                   /* device the sonode represents */
    struct vnode *so_accessvp;      /* vnode for the /dev entry */
    kmutex_t so_lock;              /* protects sonode fields */
    uint_t so_state;                /* internal state flags SS_*, below */
    uint_t so_mode;                 /* characteristics on socket. SM_* */
    mblk_t *so_ack_mp;              /* TPI ack received from below */
    mblk_t *so_conn_ind_head;       /* b_next list of T_CONN_IND */
    mblk_t *so_conn_ind_tail;
    mblk_t *so_discon_ind_mp;       /* T_DISCON_IND received from below */
    mblk_t *so_unbind_mp;          /* Preallocated T_UNBIND_REQ message */

    [...]

    int so_count;                   /* count of opened references */
    short so_family;
    short so_type;
    short so_protocol;
    short so_version;               /* From so_socket call */
    short so_pushcnt;               /* Number of modules above "sockmod" */
    short so_options;               /* From socket call, see socket.h */
    struct linger so_linger;        /* SO_LINGER value */
    int so_sndbuf;                  /* SO_SNDBUF value */
    int so_rcvbuf;                  /* SO_RCVBUF value */

    [...]
}
```

### 1.4.2 La structure `sockaddr`

**Adresse** : désignation spécifique permettant d'identifier les correspondants

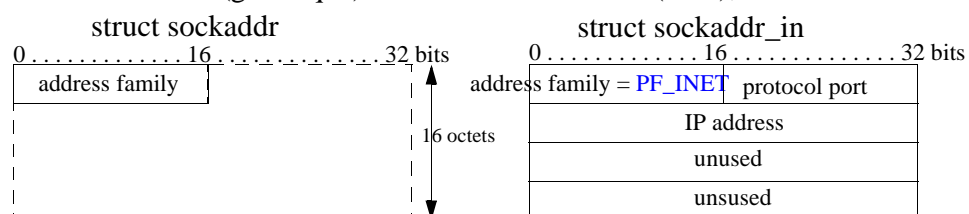
Structure décrite dans le fichier `<netinet/in.h>`

Structure polymorphe (générique) :

- adaption à la famille protocolaire utilisée :

· par exemple :

struct `sockaddr` (générique) => struct `sockaddr_un` (unix), struct `sockaddr_in` (internet).



Utilisée au sein des autres structures de données

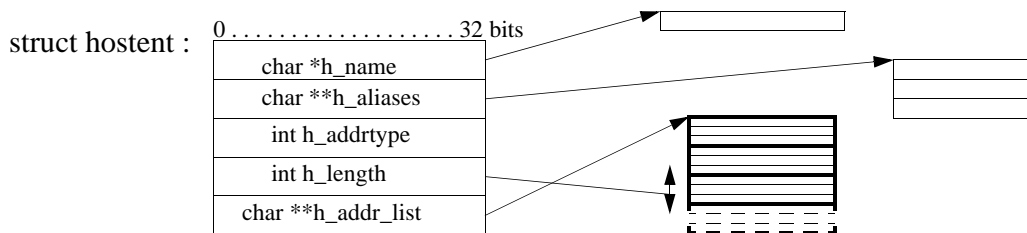
### Extrait du fichier `<netinet/in.h>`

```

/*
 * Socket address, internet style.
 */
struct sockaddr_in {
    sa_family_t    sin_family;
    in_port_t      sin_port;
    struct in_addr sin_addr;
    unsigned char  sin_zero[8];
}
    
```

### 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 (alias),
- liste d'adresses,
- type et longueur des adresses;

Informations :

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

Manipulée par les primitives : `gethostbyname()`, `gethostbyaddr()`, `gethostent()`, etc.

### Extrait du fichier `<netdb.h>`

```

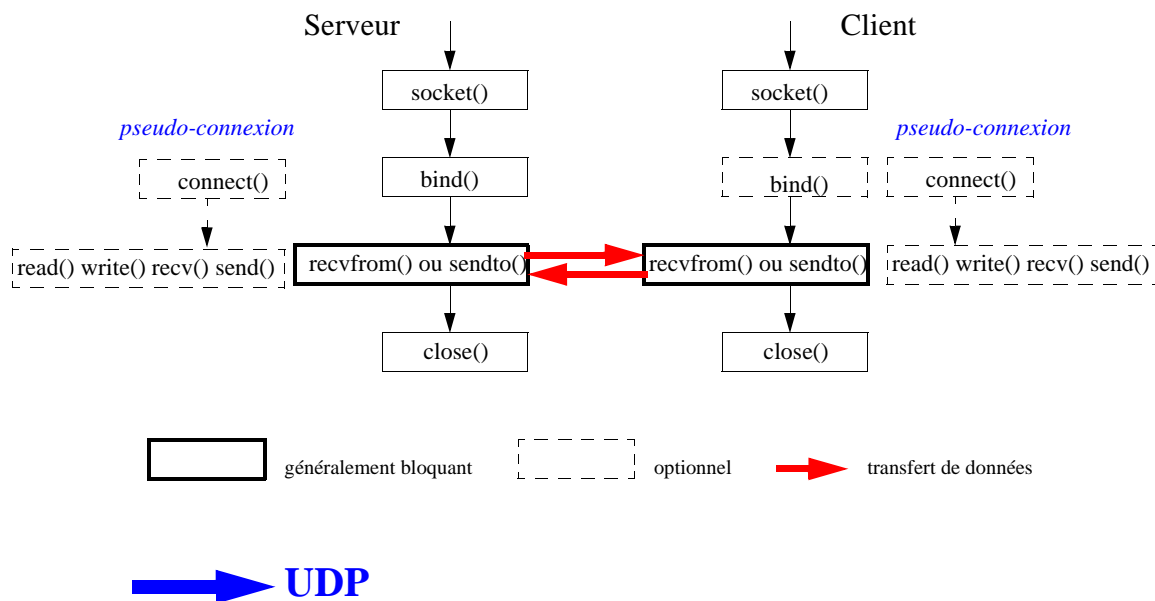
/*
 * Structures returned by network data base library.
 * All addresses are supplied in host order, and
 * returned in network order (suitable for use in system calls).
 */
struct hostent {
    char *h_name; /* official name of host */
    char **h_aliases; /* alias list */
    int h_addrtype; /* host address type */
    int h_length; /* length of address */
    char **h_addr_list; /* list of addresses from name server */
#define h_addr h_addr_list[0] /* address, for backward compatibility */
};

struct protoent {
    char *p_name; /* official protocol name */
    char **p_aliases; /* alias list */
    int p_proto; /* protocol # */
};

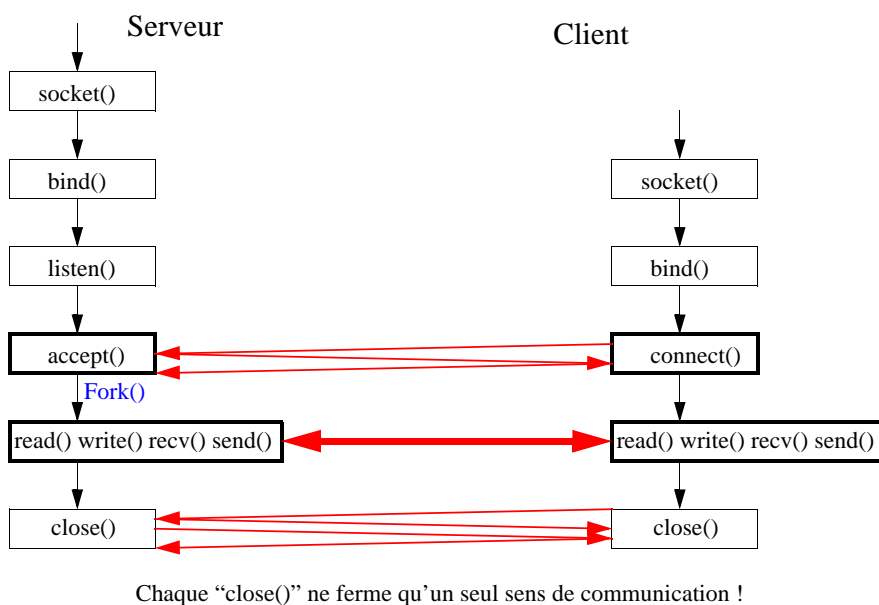
struct servent {
    char *s_name; /* official service name */
    char **s_aliases; /* alias list */
    int s_port; /* port # */
    char *s_proto; /* protocol to use */
};
    
```

## 2. L'enchaînement des primitives

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



### 2.2. Exemple d'enchaînement simple en mode connecté



**→ 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, SOCK\_RAW, 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 :

```
#include <sys/socket,h>
int s;
s=socket(PF_INET,SOCK_DGRAM,0);
```

#### 3.2. Attachement de la socket avec les informations locales

`int bind(int s, struct sockaddr * sock_addr, int lg_sockaddr);`

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>**

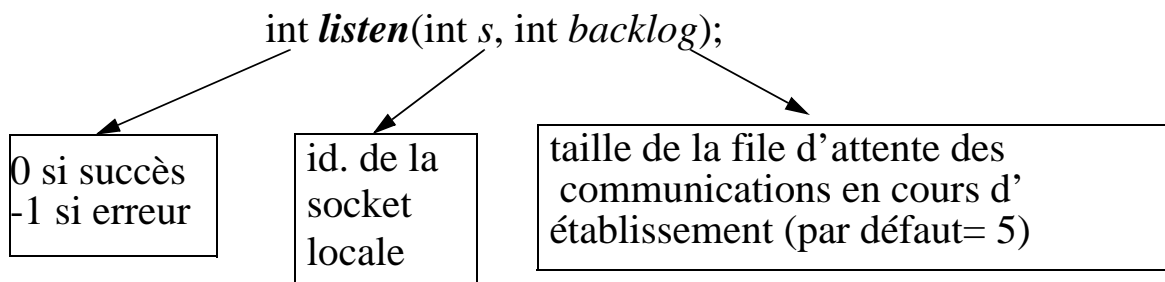
taille de la structure précédente (obtenue avec sizeof)

si le numéro de port n'est pas précisé (=0), il est choisi par le noyau

Exemple:

```
struct sockaddr_in sock_addr;
bzero((char *) &sock_addr, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;
sock_addr.sin_port = 0;
if (bind(s, (struct sockaddr *) &sock_addr, sizeof(sock_addr)) < 0) {
    perror("Echec du bind()");
    exit(ERR_BIND);
}
```

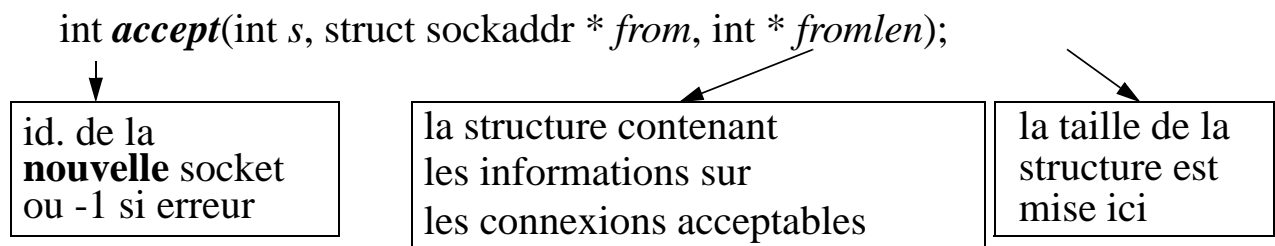
### 3.3. Longueur maximum de la file d'attente des communications en cours d'établissement



Exemple :

```
if (listen(sock,12) < 0) {
    perror("Echec du listen()");
    exit(ERR_LISTEN);
}
```

### Définition de l'ensemble des connexions acceptables



INADDR\_ANY: équivalent à n'importe quelle adresse

Exemple :

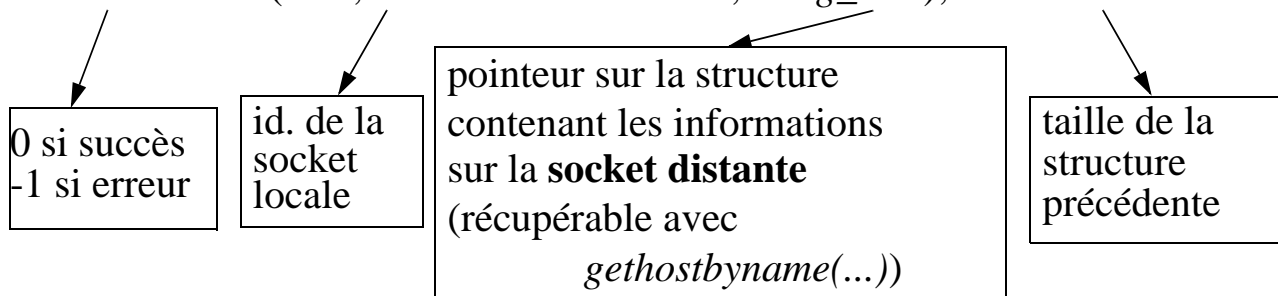
```
struct sockaddr_in from;
int from_lg;
from_lg=sizeof(from);
bzero((char *) &from, from_lg);
new_sock = accept(s, (struct sockaddr *) &from, &from_lg) {
    perror("Echec de l'accept()");
    exit(ERR_ACCEPT);
}
```



### 3.4. Etablissement de la connexion

Etablissement de la (pseudo-)connexion

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



Exemple :

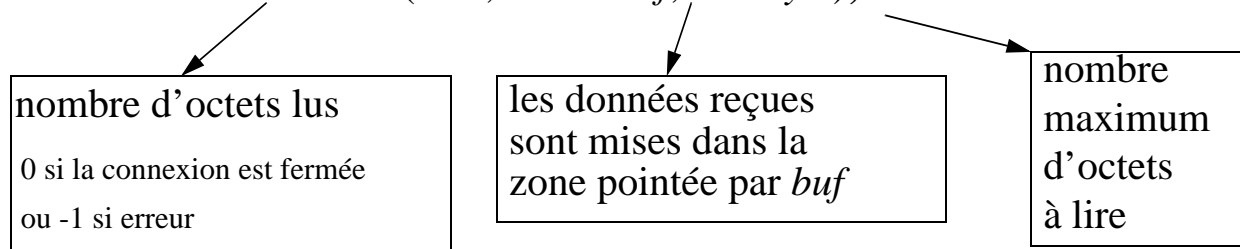
```

struct sockaddr_in dest;
    bzero((char *) &dest, sizeof(dest));
    dest.sin_family = AF_INET;
    dest.sin_port =3333;
    bcopy((char *)station_distante->h_addr_list[0], (char *) &dest.sin_addr, station_distante->h_length);
    if (connect(s, (struct sockaddr *) &dest, sizeof(dest)) < 0) {
        perror("Echec du connect()");
        exit(ERR_CONNECT);
    }
    
```

### 3.5. Réception de données

La primitive habituelle d'E/S :

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



Exemple :

```

#define MAXBUFFER 1024
char buf[MAXBUFFER];
int nb_lu, newsock;
    if ((nb_lu=read(new_sock, buf, MAXBUFFER)) <0) {
        perror("Echec en réception de données");
        exit(ERR_READ);
    }
    
```

### 3.6. Autres primitives de réception de données

Similaire au “read()” :

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

=> **flag** : MSG\_PEEK(lecture non destructive), MSG\_OOB (out of band data),  
MSG\_WAITALL(attente bloquante du nombre précis d’octets), ...

```
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, peut-être utilisé de manière sélective

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

=>envoi d’un seul message formé de plusieurs blocs de données

### 3.7. Emission de données

La primitive d’E/S habituelle, symétriques des “read, recv, recvfrom, recvmsg” précédents :

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

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

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

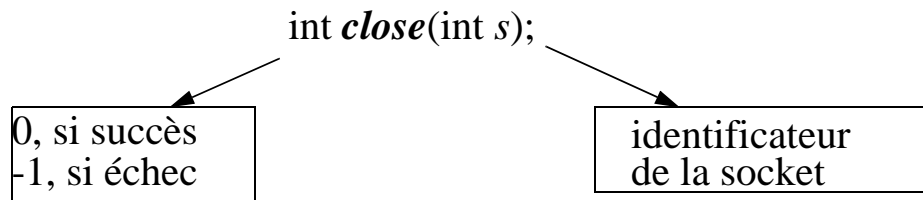
```
int sendmsg(int s, struct msghdr * msg, int flag);
```

### 3.8. Libération de la socket

La primitive de fermeture habituelle,

Libère l'espace de stockage associé à la socket,

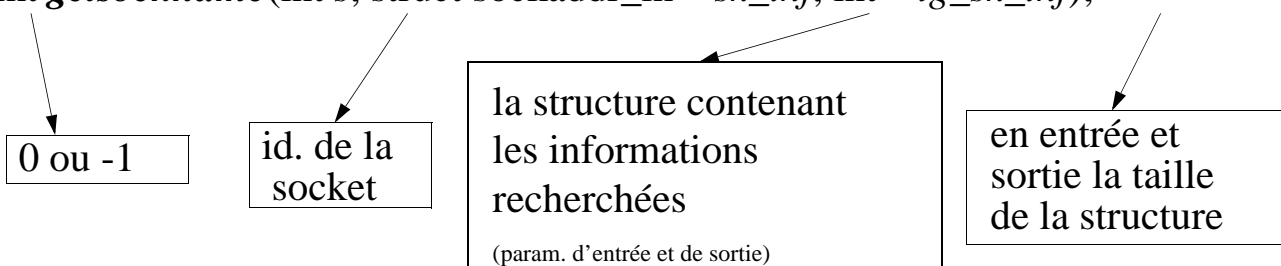
Libère la connexion (si elle existe !)



### 3.9. Accès aux informations sur une 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)*

### 3.10. Primitives d'interrogation de la base de données relatives aux stations

Obtenir des informations à partir du nom d'une station :

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

nom<sub>s</sub> et adresse<sub>s</sub> de la station dont le nom est passé en paramètre

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

Obtenir le nom de la station locale :

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

0 ou -1

le nom de la station locale (tronqué à namelen car.) est renvoyé dans name

### 3.11. Autres primitives d'interrogation :

- pour obtenir des **infos sur une machine** dont on connaît l'adresse :
  - . `gethostbyaddr()`
- pour parcourir la Bdd
  - . `sethostent()`, `gethostent()`, `endhostent()`
- pour connaître le **protocole** utilisé :
  - . `getprotobyname()`, `getprotobynumber()`...
- 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`)

Note : 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
  - . extension des sockets
  - . proposé pour IBM PC
- TLI (“transport level interface”)
  - . proposé par Unix System V
  - . implémentation par Stream (empilement de contrôleurs spécifiques)
  - . compatible avec les sockets

Pour d’autres informations :

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

## 5. Encore plus d’informations

### 5.1. L’huissier

Le processus “**inetd**” (super-serveur ou serveur de serveurs) lance, lorsque cela est nécessaire, les serveurs définis dans le fichier “/etc/inetd.conf” pour rendre les services annoncés par le fichier “/etc/services”.

==> minimise la charge du processeur

### 5.2. Les signaux

Les signaux suivants sont associés aux évènements suivants :

- **SIGIO** : la socket est prête pour une entrée-sortie
- **SIGURG** : des données urgentes sont disponibles dans une socket
- **SIGPIPE** : il n’est plus possible d’écrire sur une socket

### 5.3. Les transmissions asynchrones

Un processus peut réagir à de multiples évènements en utilisant des transmissions dites asynchrones. Il utilise alors les primitives :

- **select()** : mise en attente d’évènements multiples
- émissions ou réceptions non-bloquantes, configurables grâce aux fonctions de contrôle

#### 5.4. Les fonctions de contrôle

Utilisation non-standard des E/S

Primitives habituelles de contrôle des primitives d'E/S :

- ioctl() : paramètre = FIONBIO
- fcntl() : paramètres = O\_NDELAY(mode non-bloquant), F\_GETOWN, F\_SETOWN

Primitives réservées au contrôle des sockets :

- [getsockopt\(\)](#), [setsockopt\(\)](#) : lecture, écriture des paramètres
- paramètres : SO\_BROADCAST, SO\_REUSEADDR, SO\_KEEPALIVE, SO\_LINGER(close bloquant), SO\_SNDBUF et SO\_RCVBUF (taille des tampons d'émission et de réception)

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

- address = inet\_addr(string);
- string = inet\_ntoa(inet\_addr);

#### 5.6. Représentation des données

Il existe une représentation standard définie pour Internet :

- l'"network byte order"
- chaque système peut avoir une représentation interne différente : l'"host byte order"

Un ensemble de primitives de conversion :

- htonl()/ntohl()
- htons()/ntohs() ...
- etc

Technique inadaptée pour les autres types de données et les structures de données complexes :

⇒ RPCgen et XDR (ou ASN-1 de l'OSI)