I. Les dictionnaires

Dictionnaire des machines : Correspondance nom et adresse d'hôtes

Voici la structure que l'on récupère lorsqu'on cherche à identifier un hôte : La structure *hostent* est définie ainsi dans <*netdb.h*> :

```
struct hostent {
   char    *h_name;      // Nom officiel de l'hôte.
   char   **h_aliases;   // Table d'alias terminé par zéro, d'alternatives au nom officiel de l'hôte.
   int      h_addrtype;  // Type d'adresse de l'hôte (actuellement AF_INET)
   int      h_length;    // Longueur en octet de l'adresse
   char   **h_addr_list; // Une table, terminée par zéro, d'adresses réseau pour l'hôte,
                         //avec l'ordre des octets du réseau
};
```

```
#define h_addr h_addr_list[0]  /* adresse, pour compatibilité */
```

h_addr La première adresse dans *h_addr_list* pour respecter la compatibilité ascendante.

Elle est décrite dans /usr/include/netdb.h. Un ensemble d'appels divers existe pour récupérer une structure de ce type concernant un hôte. Par exemple gethostbyname(char *nom) où nom est le nom du hôte.

Voici le début du manuel pour cet appel :

NAME
gethostent, gethostbyaddr, gethostbyname, sethostent,
endhostent - get network host entry

```
SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
struct hostent *gethostent();
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr, int len, int type);
```

Dictionnaire des services : La structure *servent* est définie dans <*netdb.h*>

```
struct servent
{
  char  *s_name;    // Nom officiel du service
  char **s_aliases; // Liste d'alias terminée par zéro contenant d'autres
                    // noms utilisables pour le service
  int    s_port;    // Le numéro de port, donné dans l'ordre des octets du réseau
  char  *s_proto;   // Le nom du protocole utilisé par ce service
}
struct servent *getservent (void);
```

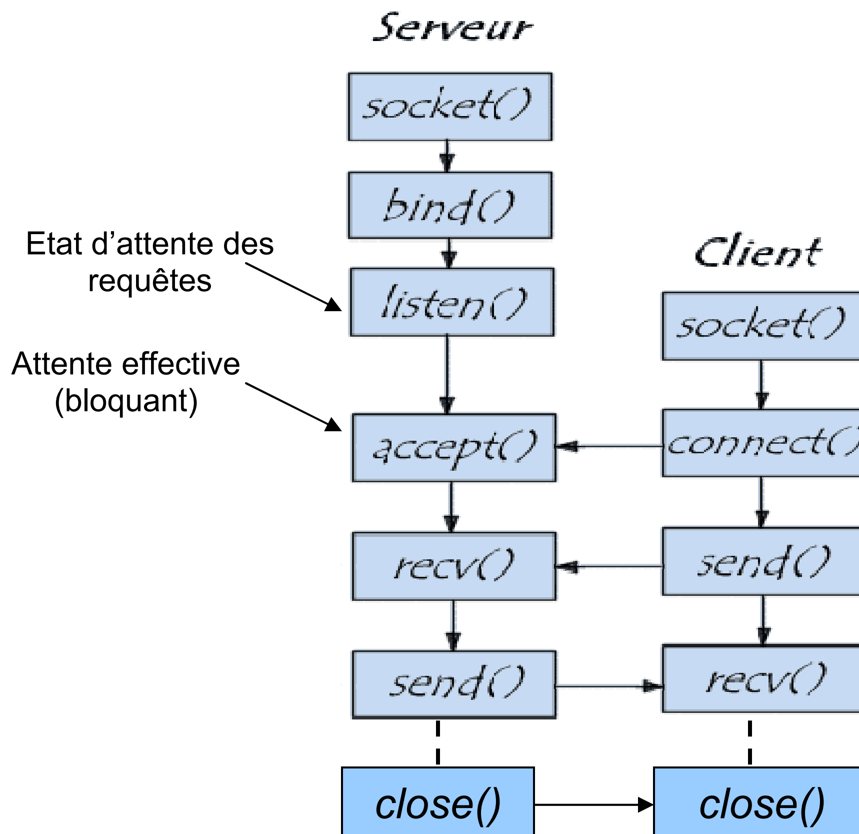
La structure *netent* est définie dans <*netdb.h*> ainsi :

```
struct netent
{
  char            *n_name;       // Nom officiel du réseau
  char           **n_aliases;    // Liste d'alias d'alternatives au nom du   réseau, terminée par zéro
  int             n_addrtype;    // Le type d'adresse du réseau, actuellement il vaut toujours AF_INET
  unsigned long int n_net; // L'adresse du réseau, utilisant l'ordre des octets de l'hôte
}
```

Dictionnaire des protocoles : La structure *protoent* est définie dans <*netdb.h*>

```
struct protoent
{
  char   *p_name;    // Le nom officiel du protocole
  char  **p_aliases; // Une liste, terminée par zéro, d'autres noms pour le protocole.
  int     p_proto;   // Numéro du protocole
}
```

II. Schéma d'une communication de type client-serveur en mode connecté (TCP)



Introduction
Il existe des sockets Internet et des sockets système (LINUX)

On s'intéresse uniquement au sockets Internet et plus précisément à 2 d'entre elle.

SOCK_STREAM : *Stream Sockets* (les socket de flux)
SOCK_DGRAM : *Datagram Sockets* (les sockets de paquets)

Les sockets de flux sont deux voies de communications bi-directionnelles et fiables (Telenet, WWW) stream sockets utilisent le protocole appelé "The Transmission Control Protocol *TCP*" et le protocole "*Internet Protocol* IP" qui traite uniquement le routage Internet.

Les sockets de paquets utilisent le protocole IP pour le routage, mais elles n'utilise et le protocole UDP (User Datagram Protocol *UDP*) (tftp,..)

Données et structures de données utilisées par les interfaces de socket

Un descripteur de socket est du type entier : int

Il existe deux façons d'arranger les octets:

Octet de poids fort en premier : "Ordre d'Octets Réseau" (*NDT:"Network Byte Order"* NBO).
Octet de poids faible en premier : "Ordre d'Octets Hôte" (*NDT:"Host Byte Order"* HBO).

Quelques conversion :

- htons()--"Host to Network Short"
- htonl()--"Host to Network Long"
- ntohs()--"Network to Host Short"
- ntohl()--"Network to Host Long"

Quand quelque chose doit être dans l'ordre *NBO* vous aurez à appeler une fonction (htons()) pour le transformer

struct sockaddr : cette structure contient les informations d'adresse de socket pour beaucoup de types de sockets:

```
struct sockaddr {
    unsigned short    sa_family;    /* famille d'adresse, AF_xxx       */
    char              sa_data[14];  /* 14 octets d'adresse de protocole */
};
```

sa_family : peut être beaucoup de choses, mais ce sera "AF_INET"
sa_data : contient une adresse de destination et un numéro de port pour la socket.

Pour utiliser la **struct sockaddr**, les développeurs ont créé une structure parallèle: **struct sockaddr_in** ("in" pour "Internet".)

```
struct sockaddr_in {
   short int          sin_family;      /* Famille d'adresse              */
   unsigned short int sin_port;        /* Numéro de Port                 */
   struct in_addr     sin_addr;        /* Adresse IP de la machine       */
   unsigned char      sin_zero[8];  /* Complément pour avoir la même taille que struct sockaddr */
};
```

Noter que **sin_zero** (pour compléter la structure à la longueur d'une **struct sockaddr**) doit être initialiser avec des zéros à l'aide des fonctions **bzero()** ou **memset()**.

Un pointeur vers une **struct sockaddr_in** qui peut être instancié (**cast**) en un pointeur vers une **struct sockaddr** et vice-versa.

sin_port et **sin_addr** doivent être en **Network Byte Order**

```
/* Internet adresse (une structure pour des raisons historique) */
struct in_addr {
   unsigned long s_addr;
};
```

1. La fonction socket () (définition d'une socket)

l'appel système socket() pour obtenir un descripteur de socket, voici le détail:

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

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

domain :
AF_UNIX : local au système, nom de fichier dans l'arborescence
AF_INET : accès au réseau en utilisant le protocole IP
AF_ROUTE : passage de paramètres au noyau (table de routage, ARPT)
type :
SOCK_STREAM : protocole de type mode connecté (TCP)
SOCK_GRAM : protocole de type mode datagramme (UDP)
SOCK_ROW : utilisation directe de protocole de bas niveau 3 (IP/ICMP)
protocol : identification du protocole utilisé
si 0, le système déduit ce champ des deux paramètres précédents

2. La fonction bind() (lien entre la socket et le protocole : adresse IP et numéro de service)

Après création du socket, il s'agit de le lier à un point de communication défini par une adresse et un port, c'est le rôle de la fonction *bind()* :

int bind(int descripteur, sockaddr localaddr, int addrlen)

- **valeur retour** : entier qui contient le compte-rendu de l'instruction
  » 0 : opération correctement déroulée
  » -1 : une erreur est survenue
- **descripteur** représente le descripteur du socket nouvellement créé
- **localaddr** est une structure qui spécifie l'adresse locale à travers laquelle le programme doit communiquer

3. La fonction listen()

La fonction *listen()* permet de mettre un socket en attente de connexion(création de file d'attente), elle ne s'utilise qu'n mode connecté (donc avec le protocole TCP)

int listen(int socket, int backlog)

socket représente le socket précédemment ouvert
backlog représente le nombre maximal de connexions pouvant être mises en attente

La fonction listen() retourne la valeur SOCKET_ERROR en cas de problème, sinon elle retourne 0.

Voici un exemple d'utilisation de la fonction listen() :
if (listen(socket,10) == SOCKET_ERROR) {   // *traitement de l'erreur* }

4. La fonction accept()

La fonction *accept()* permet la connexion en acceptant un appel :

int accept(int socket, struct sockaddr * addr, int * addrlen)

- **socket** représente le socket précédemment ouvert (le socket local)
- **addr** représente un tampon destiné à stocker l'adresse de l'appelant
- **addrlen** représente la taille de l'adresse de l'appelant

La fonction *accept()* retourne un identificateur du socket de réponse. Si une erreur intervient la fonction *accept()* retourne la valeur *INVALID_SOCKET*.

Voici un exemple d'utilisation de la fonction *accept()* :

Sockaddr_in Appelant;

/* structure destinée à recueillir les renseignements sur l'appelant
Appelantlen = sizeof(from);

accept(socket_local, (struct sockaddr*)&Appelant, &Appelantlen);

5. La fonction connect()

La fonction *connect()* permet à un client d'établir une connexion avec le serveur : bloquant

int connect(int socket,struct sockaddr * addr,int * addrlen)

- **valeur retour** : entier qui contient le compte-rendu de l'instruction
  » 0 : opération correctement déroulée
  » -1 : une erreur est survenue
- **socket** représente le socket précédemment ouvert (le socket à utiliser)
- **addr** représente l'adresse de l'hôte à contacter. Pour établir une connexion, le client ne nécessite pas de faire un *bind()*
- **addrlen** représente la taille de l'adresse de l'hôte à contacter

Voici un exemple d'utilisation de la fonction *connect()*, qui connecte le socket "s" du client sur le port *port* de l'hôte portant le nom *serveur* :

```
toinfo = gethostbyname(serveur);
toaddr = (u_long *)toinfo.h_addr_list[0];

/* Protocole internet */
to.sin_family = AF_INET;

/* Toutes les adresses IP de la station */
to.sin_addr.s_addr = toaddr;

/* port d'écoute par défaut au-dessus des ports réservés */
to.sin_port = htonl(port);

if (connect(socket,(struct sockaddr*)to,sizeof(to)) == -1) {
// Traitement de l'erreur;
}
```

6. La fonction recv()

La fonction *recv()* permet de lire dans un socket en mode connecté (TCP) :

int recv(int socket,char * buffer,int len,int flags)

- **socket** représente le socket précédemment ouvert
- **buffer** représente un tampon qui recevra les octets en provenance du client
- **len** indique le nombre d'octets à lire
- **flags** correspond au type de lecture à adopter :
  - le flag *MSG_PEEK* indiquera que les données lues ne sont pas retirées de la queue de réception
  - le flag *MSG_OOB* indiquera que les données urgentes (*Out Of Band*) doivent être lues
  - le flag *0* indique une lecture normale

La fonction *recv()* renvoie le nombre d'octets lus. De plus cette fonction bloque le processus jusqu'à ce qu'elle reçoive des données.

Voici un exemple d'utilisation de la fonction *recv()* :

```
retour = recv(socket,Buffer,sizeof(Buffer),0 );
if (retour == SOCKET_ERROR) {   // traitement de l'erreur }
```

7. La fonction send()

La fonction *send()* permet d'écrire dans un socket (envoyer des données) en mode connecté (TCP) :

int send(int socket,char * buffer,int len,int flags)

- **socket** représente le socket précédemment ouvert
- **buffer** représente un tampon contenant les octets à envoyer au client
- **len** indique le nombre d'octets à envoyer
- **flags** correspond au type d'envoi à adopter :
  - le flag *MSG_DONTROUTE* indiquera que les données ne routeront pas
  - le flag *MSG_OOB* indiquera que les données urgentes (*Out Of Band*) doivent être envoyées
  - le flag *0* indique un envoi normal

La fonction *send()* renvoie le nombre d'octets effectivement envoyés.

Voici un exemple d'utilisation de la fonction *send()* :

```
retour = send(socket,Buffer, sizeof(Buffer), 0 );
if (retour == SOCKET_ERROR) {    // traitement de l'erreur  }
```

8. Conversion d'un fichier non bufférisé en fichier bufférisé (très conseillé !!!)

```
FILE * DescFichierBufferise;

theIn =fdopen(theConversion, "r ou w"); //en lecture ou ecriture
if(theIn == NULL) { traitement();}

close(0);  dup(theConversion);
close(1);  dup(theConversion);
```

<u>Lecture par ligne</u> : fgets ou gets

fgets (*chaîne* , *taille* , *flot-de-données*)   ou gets(*chaîne* , *taille*)

- *chaîne* est de type pointeur vers char et doit pointer vers un tableau de caractères.
- *taille* est la taille en octets du tableau de caractères pointé par *chaîne*.
- *flot-de-données* est de type pointeur vers FILE. Il pointe vers le fichier à partir duquel se fait la lecture.

```
#include <stdio.h>
#define LONG 128
char ligne[LONG];
FILE *fi;

while (fgets(ligne,LONG,fi) != NULL)   /*  stop sur fin de fichier ou erreur  */
   {
   ...   /*   utilisation de ligne   */
   }
```

valeur rendue
La fonction fgets rend le pointeur *chaîne* cas de lecture sans erreur, ou NULL dans le cas de fin de fichier ou d'erreur.

<u>Ecriture par ligne</u> : fputs ou puts

fputs (*chaîne* , *flot-de-données*) ou fputs (*chaîne*)

- *chaîne* est de type pointeur vers char. Pointe vers un tableau de caractères contenant une chaîne se terminant par un *null*.
- *flot-de-données* est de type pointeur vers FILE. Il pointe vers le fichier sur lequel se fait l'écriture

```
#include <stdio.h>
#define LONG ...
char ligne[LONG];
FILE *fo;

fputs(ligne,fo);
```

Valeur rendue
La fonction fputs rend une valeur non négative si l'écriture se passe sans erreur, et EOF en cas d'erreur.

9. Les fonctions close() et shutdown()

La fonction *close()* permet la fermeture d'un socket en permettant au système d'envoyer les données restantes (pour TCP) :

int close(int socket)

La fonction *shutdown()* permet la fermeture d'un socket dans un des deux sens (pour une connexion full-duplex) :

int shutdown(int socket, int how)

- Si *how* est égal à 0, le socket est fermé en réception
- Si *how* est égal à 1, le socket est fermé en émission
- Si *how* est égal à 2, le socket est fermé dans les deux sens

close() comme *shutdown()* retournent -1 en cas d'erreur, 0 si la fermeture se déroule bien.