Communication dans le domaine Internet sous Unix à l'aide de sockets

(Z. Mammeri – UPS)

1. Introduction

Une **socket** est un point de communication par lequel un processus peut émettre ou recevoir des informations à partir d'un autre processus se trouvant sur la même machine ou sur une machine distante (qui appartient ou non au même réseau local).
Une socket est identifiée via un **descripteur** de socket de même nature que ceux qui identifient les fichiers dans un système Unix. Il est en particulier possible de rediriger les fichiers standards (descripteurs 0, 1 et 2) sur des sockets.
Un descripteur de socket est connu du processus créateur de la socket et de tous ses fils qui ont été créés après la création de la socket.
Pour utiliser les sockets, il faire inclure dans les programmes le fichier **`<sys/socket.h>`**.

2. Caractéristiques d'une socket

a) Domaine d'une socket

Le domaine d'une socket spécifie le format des adresses qui pourront être données à la socket et les différents protocoles supportés pour la communication.
Il existe deux principaux domaines de sockets : le domaine Unix (`AF_UNIX`) et le domaine Internet (**`AF_INET`**). Nous nous intéressons ici au domaine Internet seulement. Pour le domaine Internet, les adresses de sockets sont définies à l'aide de la structure `sockaddr_in`.

```
struct sockaddr_in {
      short sin_family;       /* Famille de l'adresse = AF_INET*/
      u_short sin_port;       /* numéro de port */
      struct in_addr sin_addr;/* Adresse Internet */
      char sin_zero[8];       /* Champ de 8 zéros */
                    }
```

Le champ de huit zéros sert à faire coïncider la taille de la structure `sockaddr_in` avec la structure générique `sockaddr.`

b) Type d'une socket

Les types de sockets autorisés sont les suivants :

- Type `SOCK_DGRAM` : c'est un mode de communication non connecté pour l'envoi de datagrammes de taille bornée. Le protocole sous-jacent est UDP.
- Type `SOCK_STREAM` : c'est un mode de communication en mode connecté et fiable. Les données sont délivrées dans l'ordre de leur émission. Le protocole sous-jacent est TCP.
- Types `SOCK_RAW, SOCK_SEQPACKET` : peu utilisés et ne seront pas développés ici.

3. Création et suppression d'une socket

La création d'une socket se fait à l'aide la primitive **socket** suivante :

```
int socket(domaine, type, protocole)
      int domaine;      /* sa valeur sera limitée ici à AF_INET */
      int type;         /* sa valeur sera limitée ici à SOCK_DGRAM ou
                           SOCK_STREAM */
      int protocole;    /* sa valeur sera limitée ici à 0 */
```

La valeur rendue est -1 (en cas d'échec de la fonction) ou la valeur du descripteur de la socket créée.

La fermeture d'une socket se fait via la fonction **close**.

4. Attachement d'une socket à une adresse

Après sa création, une socket n'est accessible que par les processus qui en connaissent le descripteur (il s'agit du processus créateur de la socket et de ses fils créés après la socket). Pour permettre à des processus n'ayant pas de descripteur de la socket d'utiliser la socket, un mécanisme de nommage de socket peut être utilisé. Ce mécanisme (fourni par la fonction **bind**) permet de nommer une socket. En utilisant le nom de la socket, des processus peuvent émettre ou recevoir des informations via cette socket.

```
int bind(sock, p_adresse, lg)
      int sock;         /* descripteur de socket */
      struct sockaddr_in *p_adresse;     /* pointeur sur l'adresse */
      int lg;           /* longueur de l'adresse */
```

La valeur rendue est -1 (en cas d'échec) et 0 (en cas de succès).

Le paramètre `p_adresse` contient les informations permettant de désigner la socket. Tout processus qui connaît `p_adresse` peut utiliser la socket.

L'attachement d'une adresse Internet à une socket nécessite la préparation d'un objet ayant la structure `sockaddr_in`. Cela suppose :
- la connaissance de l'adresse de la machine locale qui peut être obtenue via la primitive `gesthostname` et la fonction `gethostbyname`).
- le choix d'un numéro de port. On peut choisir un port d'un service existant à condition d'avoir des droits de l'utiliser, ou d'un numéro de port quelconque choisi par l'utilisateur ou laisser à l'initiative du système en mettant le champ `sin_port` à zéro (cette dernière possibilité est conseillée pour sa simplicité).

La fonction **`getsockname`** permet de récupérer l'adresse d'une socket dont le processus dispose d'un descripteur.

```
int getsockname (sock, p_adr, p_lg)
      int sock          /* descripteur de la socket */
      struct sockaddr *p_adr;      /* pointeur pour l'adresse */
      int *p_lg;        /* pointeur sur la longueur de l'adresse */
```

Les paramètres `p_adr` et `p_lg` permettent de récupérer l'adresse et la longueur de cette adresse.
La fonction `bind` rend - 1 en cas d'erreur et 0 sinon.

Exemple :

```
/*
  creersock : fonction de création et de nommage d'une socket.
  Le numéro de port souhaité est donné en paramètre.
  Le descripteur de socket est renvoyé en résultat.
  Si le paramètre d'entrée port est nul, le numéro de port effectivement choisi pour la socket est renvoyé dans le
paramètre port.

  Ce programme est inspiré de l'exemple donné dans " J.-M. Rifflet, La Communication sous Unix, page 250 "*/

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

int creersock(port,type)
int *port;
int type;
{  int i;
    int desc;                       /* descripteur de la socket */
    struct sockaddr_in nom;         /* adresse de la socket */
    int longueur;                   /* longueur de l'adresse */

    /* Création de la socket */
if((desc=socket(AF_INET,type,0))==-1){
            printf("\n création de socket impossible");
            return(2);                  }

   /* Préparation de l'adresse */
    bzero((char *)&nom, sizeof(nom));
    nom.sin_port= *port;
    nom.sin_addr.s_addr = INADDR_ANY;
    nom.sin_family = AF_INET;
    i=bind(desc,&nom,sizeof(nom));
    if(i==-1)    {
            printf("\n Nommage de socket impossible >>> code erreur = %d \n",i);
            return(3); }
     printf("\n La socket a été créée et attachée \n");

   longueur=sizeof(nom);
   if(getsockname(desc,&nom,&longueur)==-1)
            {printf("\n Erreur d'accès aux attributs de la socket \n");}
    *port=ntohs(nom.sin_port);
    return(desc);
}
main()
{int port=0;
 int type = SOCK_STREAM;
 creersock(&port,type);
 printf("\n Numéro du port retenu %d \n", port);
}
```

5. Programmation en mode connecté (par des SOCK_STREAM)

Ce mode est utilisé par beaucoup d'applications Internet (ftp, telnet, ...). C'est un mode de communication fiable (la fiabilité se paie par une certaine lourdeur et un accroissement du volume de données).

Deux entités distantes qui souhaitent dialoguer doivent établir une connexion entre elles. Nous appelons processus *initiateur* (ou client) d'une connexion, l'entité qui demande l'établissement d'une connexion. L'entité qui accepte la connexion est appelée processus *correspondant* (ou serveur).

5.1. Fonctionnement du processus initiateur de la connexion

Comme son nom l'indique c'est ce processus qui provoque l'établissement de la connexion. L'autre processus (le correspondant) est passif et attend que l'initiateur demande d'établissement de connexion. La demande d'établissement de la connexion s'effectue via la primitive **connect** qui conduit à la création d'un circuit virtuel entre les deux interlocuteurs (le processus_initiateur et le processus_correspondant). Ce circuit virtuel permet des échanges bidirectionnels.

```
int connect (sock, p_adr, lgadr)
      int sock;              /* descripteur de la socket locale */
      struct sockaddr *p_adr; /* adresse de la socket distante */
      int lgadr;             /* Longueur de l'adresse distante */
```

La socket correspondant au descripteur `sock` sera, dans le cas où elle ne l'a pas déjà été (via la fonction `bind`), attachée automatiquement à une adresse locale.

Condition nécessaire pour la réussite de l'opération de connexion : le paramètre `*p_adr` est associé à une socket derrière laquelle un processus_correspondant a exécuté une opération `listen` (`listen` est expliquée par la suite). `*p_adr` ne doit pas être déjà utilisé pour une autre connexion.
En cas de réussite, la fonction `connect` renvoie 0. En cas d'échec, elle renvoie -1.

5.2. Fonctionnement du processus_correspondant

Le fonctionnement du processus correspondant se déroule selon les phases suivantes :

a) Création et attachement d'une socket d'écoute (en utilisant les fonctions *socket* et *bind*)

L'interlocuteur doit disposer d'une socket d'écoute qui lui permet d'être avisé quand une demande de connexion est émise par l'initiateur d'une connexion. Lorsque la demande de connexion arrive, le processus est réveillé. La requête de connexion peut être traitée par le processus lui-même ou par un fils créé à cet effet.

b) Acceptation de connexions

Les demandes de connexion sont stockées dans une file associée à la socket d'écoute. Le processus_correspondant peut les prendre en compte ou non. Le processus_correspondant signale au système son acceptation des demandes de connexion via la primitive **listen**.

```
int listen (sock, nb)
      int sock; /* descripteur de la socket d'écoute préparée dans la
                   phase a) */
      int nb; /* nombre maximal de connexions pendantes, c.-à-d. en
        attente de traitement par le processus_correspondant */
```

La fonction `listen` renvoie -1 en cas d'échec et 0 sinon.

c) Extraction de demande de connexion

Le processus_correspondant extrait de la file des demandes de connexions pendantes une connexion pour la traiter. Lorsqu'une demande de connexion est extraite par la primitive **accept**, la connexion est établie entre le processus_initiateur et le processus_correspondant.

La liaison de la connexion avec le processus_initiateur est réalisée avec une nouvelle socket dont le descripteur est envoyé comme résultat de la fonction `accept`. Cette nouvelle socket que le système vient de créer est rattachée à un port pris parmi les ports non réservés.

On récupère, en retour de la fonction `accept`, via `p_adr` l'adresse de la socket du processus_initiateur avec laquelle la connexion a été établie.

La fonction `accept` rend -1 en cas d'erreur et le descripteur de la nouvelle socket sinon.

```
int accept (sock, p_adr, p_lgadr)
      int sock;         /* descripteur de la socket créée pour
                        dialoguer avec le processus_initiateur */
      struct sockaddr *p_adr; /* adresse de la socket connectée */
      int *p_lgadr;           /* pointeur sur la taille de la zone
                              allouée à p_adr */
```

d) Réalisation du service

Après acceptation d'une connexion, le processus_correspondant a deux possibilités :
- prendre en charge lui-même les communications avec le processus_initiateur
- ou bien sous-traiter le travail à un processus fils.

6. Echange de données

Une fois établie la connexion entre deux machines, les deux processus (initiateur et correspondant) peuvent échanger des flots de données. Une opération d'écriture dans une socket peut correspondre à une ou plusieurs opérations de lecture et inversement une opération de lecture à partir d'une socket peut correspondre à une ou plusieurs opérations d'écriture. Les différents fragments de données émis via une socket sont prélevés par leur destinataire dans l'ordre de leur émission (TCP préserve l'ordre d'émission des fragments). Par exemple, si un processus émet deux messages de 10 caractères chacun, ces deux messages peuvent être délivrés en quatre étapes, si l'autre processus lit les données émises à raison de 6 caractères par opération de lecture.

a) Emission de données

L'écriture sur une socket est réalisée via les fonctions **`write`** ou **`send`** (l'opération fournie par `write` est incluse dans celle fournie par `send`, nous présentons ici la fonction `send`).

```
int send (sock, msg, lg, option)
      int sock;   /* descripteur de la socket locale */
      char *msg;  /* adresse en mémoire du message à envoyer */
      int lg;     /* longueur du message */
      int option; /* 0, MSG_PEEK ou MSG_OOB */
```

L'utilisation de `send` avec le paramètre `option` égal à `0` est équivalente à celle de `write`. L'écriture sur une socket bloque le processus qui effectue cette opération si le tampon de réception du destinataire est plein ou bien si le tampon d'émission de la socket locale est plein.
Le nombre de caractères émis est renvoyé comme résultat. En cas d'échec, la fonction renvoie -1.

Si le paramètre `option` est égal à `MSG_OOB`, il y a demande d'émission en urgence (c'est-à-dire, demande d'émission de caractères éventuellement avant ceux qui sont en attente).

b) Réception de données

La réception (ou la lecture) à partir d'une socket est réalisée par les fonctions **`read`** ou **`recv`** (l'opération fournie par `read` est incluse dans celle fournie par `recv`, nous présentons ici la fonction `recv`).

```
int recv (sock, msg, lg, option)
      int sock;   /* descripteur de la socket locale */
      char *msg;  /* adresse en mémoire où sera stocké le message */
      int lg;     /* nombre de caractères à recevoir */
      int option; /* 0 ou MSG_OOB */
```

L'opération de réception de données est bloquante s'il n'y a pas de données au niveau de la file d'attente de réception locale. Si un processus est bloqué suite à une opération `recv`, il est débloqué dès qu'au moins un caractère est reçu.

7. Fermeture de connexion

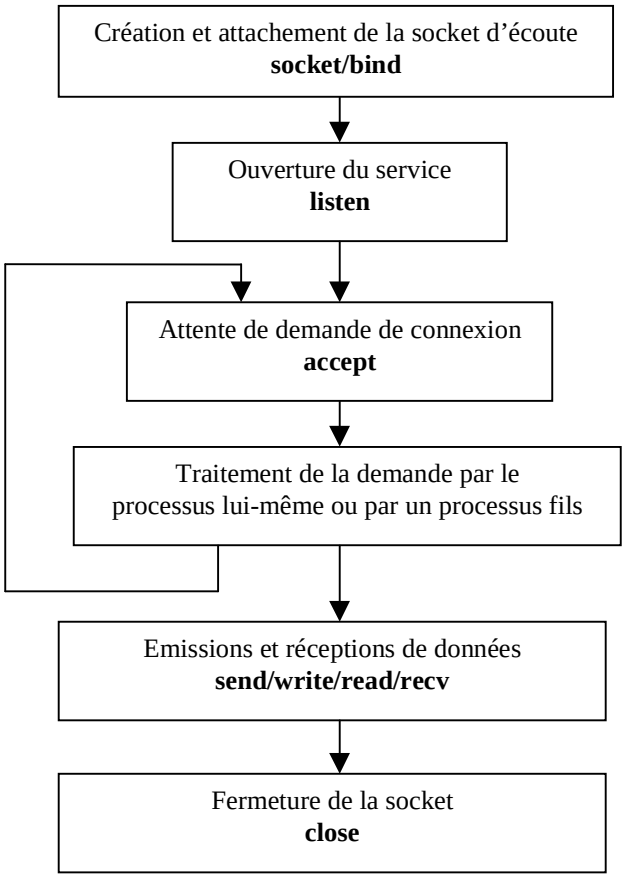
Lorsqu'un processus ne souhaite plus utiliser (dans un sens ou dans les deux sens) une connexion, il peut couper celle-ci grâce à la fonction **`shutdown`**.

```
int shutdown (sock, sens)
      int sock;   /* descripteur de la socket locale */
      int sens;
```

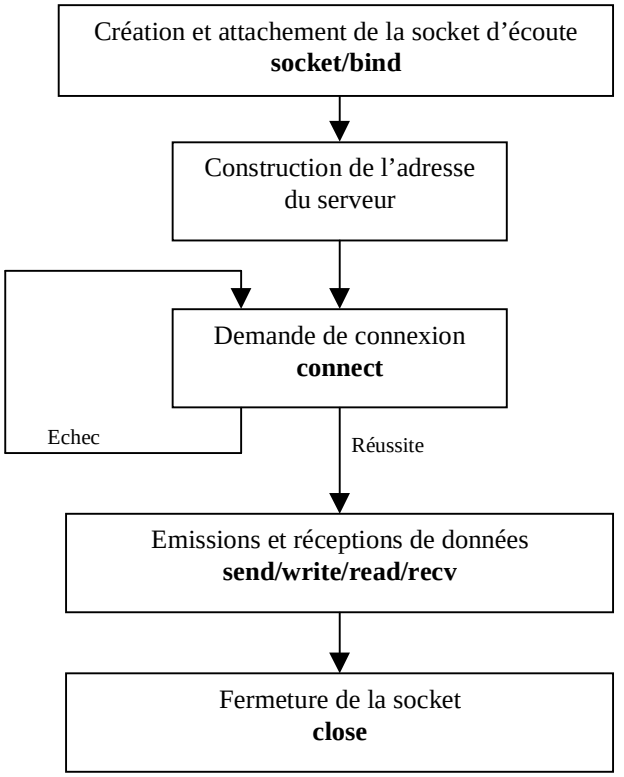
Le paramètre `sens` a la valeur 0 (si le processus ne souhaite plus recevoir), 1 (si le processus ne souhaite plus émettre) ou 2 (si le processus ne souhaite plus ni recevoir, ni émettre)

8. Remarques

- Une tentative d'écriture sur une connexion coupée provoque la réception d'un signal SIGPIPE.
- Les fonctions associées aux socket qui sont bloquantes par défaut peuvent être rendues non bloquantes via les opérations **`ioctl`** ou **`fcntl`**.
- L'asynchronisme entre les processus utilisant une connexion de sockets et le système (c'est-à-dire, l'arrivée de données sur des sockets) peut être réalisé en utilisant les signaux SIGIO et SIGURG.



Fonctionnement du serveur



Fonctionnement du client

Exemple

Application avec deux processus qui dialoguent via une connexion de sockets.

```
/* Programme du processus initiateur d'une connexion.
Le processus initiateur pose des questions auxquelles répond le processus correspondant */
/* Le numéro de port sur lequel le processus correspondant attend est égal à 1500 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <string.h>
#include <netinet/in.h>
#include <netdb.h>

int taille_maxi=100;

short int Port_ecoute=1500;

/* Fonction de création d'une socket */
int creersock()
{   int desc;                       /* descripteur de la socket */
    struct sockaddr_in nom;         /* adresse de la socket */
    int longueur;                   /* longueur de l'adresse */

  /* Création de la socket */
    if((desc=socket(AF_INET,SOCK_STREAM,0))==-1){return(-1);}
  /* Préparation de l'adresse */
    bzero((char *)&nom,sizeof(nom));
    nom.sin_port= 0;
    nom.sin_addr.s_addr = INADDR_ANY;
    nom.sin_family = AF_INET;
    if(bind(desc,&nom,sizeof(nom))==-1) return(-2);
    return(desc);
}

main()
{   char nom_machine_distante[30];
    int desc;                       /* Descripteur de la socket locale */
    struct sockaddr_in nom;         /* Adresse de la socket destinataire */
    struct hostent *hp, *gethostbyname();
    union adr {
        int adr_i;
        unsigned char adr_c[4];
            } adr, **q;

    char zone_com[100]; /* Buffer pour les communications de données */
    int lg;
    int i, bool;
    char *str_fin ="FIN";
    int lg_fin=3;

    /* Création de la socket */
    if((desc=creersock())<=-1)
            { printf("\n Erreur de création de la socket locale : erreur %d \n",desc); exit(1);}
    printf("\n Donnez le nom de la machine distante : ");
    gets(nom_machine_distante);


    /* Recherche de l'adresse Internet de la machine distante */
```

```
   if((hp=gethostbyname(nom_machine_distante))==NULL)
     {printf("\n %s : machine inconnue\n",nom_machine_distante); exit(2); }

 q= (union adr **)hp->h_addr_list;
 printf("\n Adresse distante : %u.%u.%u.%u\n",
   (*q)->adr_c[0],(*q)->adr_c[1],(*q)->adr_c[2],(*q)->adr_c[3]);


   /* Préparation de l'adresse de la socket destinataire */
   bzero((char *)&nom,sizeof(nom));
   bcopy(hp->h_addr,&nom.sin_addr,hp->h_length);
   nom.sin_family = AF_INET;
   nom.sin_port = htons(Port_ecoute);

   /* Demande de connexion */
   while (connect(desc,&nom,sizeof(nom))==-1) {perror(NULL);
    printf("\n erreur de connexion : %d\n", errno); usleep(5000000); }

   printf("\n Connexion établie \n");
   printf("\n Vous pouvez maintenant poser vos questions à votre correspond \n");
   bool=1;  /* On suppose qu'il y a au moins une question à envoyer */
   printf("\n >>>> Vous allez poser des questions à votre interlocuteur.\n");
   printf("\n >>>> S'il n'y a plus de question tapez FIN \n\n");
   while(bool==1)
    {  bzero(zone_com,taille_maxi);
       printf("\n\nQuestion : ");gets(zone_com);
       lg=strlen(zone_com);
     if (strcasecmp(zone_com,str_fin)!=0) {
              send(desc,zone_com,lg,0);
          printf("Réponse :");
              bzero(zone_com,taille_maxi);
              recv(desc,zone_com,taille_maxi,0);
              printf("%s",zone_com);
              }
       else bool=0;
    }

 send(desc,zone_com,lg_fin,0);
 printf("\n>>>>FIN du dialogue entre les deux interlocuteurs \n");

 shutdown(desc,2);
 close(desc);

}
```

```
/* Programme du processus_correspond (ou serveur) d'une connexion.
 Le processus correspondant répond aux questions posées par le processus initiateur */
/* Port d'écoute a comme numéro 1500 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>

u_short port_ecoute = 1500;
int taille_maxi=100;

/* Fonction de création d'une socket */
int creersock()
{   int desc;                       /* descripteur de la socket */
    struct sockaddr_in nom;         /* adresse de la socket */
    int longueur;                   /* longueur de l'adresse */
    u_long machine_id;
    int x;
  /* Création de la socket */
    if((desc=socket(AF_INET,SOCK_STREAM,0))==-1){return(-1);}
  /* Préparation de l'adresse */
    bzero((char *)&nom,sizeof(nom));
    nom.sin_port= htons(port_ecoute);
    nom.sin_addr.s_addr = INADDR_ANY;
    nom.sin_family = AF_INET;
    if(bind(desc,&nom,sizeof(nom))==-1) return(-2);
    return(desc);
}

main()
{
    int sock_ecoute, sock_service;
    struct sockaddr_in adr;
    int lgadr = sizeof(adr);
    int etat, bool;
    char zone_com[100]; /* Buffer pour les communications de données */
    int lg;

    /* Création de la socket d'écoute */
    sock_ecoute=creersock();
    if(sock_ecoute<=-1){
            fprintf("\n >>> Création/liaison de la socket d'écoute impossible\n");
            exit(0);
    }
    printf("\n Opération réussie pour la création de la socket d'écoute \n");
    /* Création de la file de connexions pendantes */
    if (listen(sock_ecoute,5)!=0) {printf("\n Erreur de la fonction Listen \n");
                        exit(0); }
    printf("\n Opération listen réussie \n");

    lgadr = sizeof(adr);

  /* Attente d'acceptation d'une connexion */
    printf("\n Attente de la réception de la demande de connexion \n");
    printf("\n L'opération accept est bloquante \n");
    sock_service = accept(sock_ecoute,&adr,&lgadr);

 if(fork()==0){
          bool=1; /* On suppose qu'il y a au moins une question à traiter */
```

```
            close(sock_ecoute);
        while(bool==1)
          {  bzero(zone_com,taille_maxi);
            recv(sock_service,zone_com,taille_maxi,0);
            if(strcasecmp("FIN",zone_com)==0) bool=0;
            else {
                printf("\n Question : %s",zone_com);
                bzero(zone_com,taille_maxi);
                printf("\n Votre réponse :"); gets(zone_com);
                lg=strlen(zone_com);
                    send(sock_service,zone_com,lg,0);
                }
          }
        shutdown(sock_service,2);
        close(sock_service);
            exit(0);
                }
    close(sock_service);
    close(sock_service);
    wait(&etat);
    printf("\n FIN du dialogue entre les deux interlocuteurs\n");
}
```

Fichiers, structures et fonctions pour la communication via Internet

1. Fichiers de configuration

Le fichier **`/etc/hosts`** contient les noms et numéros IP des machines du réseau local auquel appartient la machine sur laquelle se trouve ce fichier.

```
# Exemple de contenu du fichier /etc/hosts
# Sun Host Database

127.0.0.1localhost
#
#sparc10 enseignement fac de sciences
193.48.167.1    sparc10 loghost     triumph
#passerelle entree de la fac (cisco 3000)
193.48.167.2    fac-gw
#Terminaux X et PC des salles
193.48.167.3    termx3
193.48.167.4    termx4
193.48.167.60   pcst0
193.48.167.91   printpro
193.48.167.92   printmath
193.48.167.109  dao9
193.48.167.110  dao10
193.48.167.151  serv_nov
```

Le fichier **`/etc/networks`** contient les noms des réseaux connus constituant le réseau logique Internet.

```
# Exemple de contenu du fichier /etc/networks
# Sun customer networks
loopback 127
#sun-ether      192.9.200    sunether ethernet localnet
fst.univ-lehavre.fr   193.48.167   localnet
sun-oldether    125          sunoldether
#
# Internet networks
arpanet         10           arpa
ucb-ether46           ucbether
```

Le fichier **`/etc/services`** contient les noms des services Internet connus (tels que telnet, ftp, ...).

```
# Exemple de contenu du fichier /etc/services
#
tcpmux          1/tcp        # rfc-1078
discard         9/tcp        sink null
daytime         13/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp       mail
name            42/udp       nameserver
domain          53/udp
finger          79/tcp
#
# UNIX specific services
#
exec            512/tcp
login           513/tcp
shell           514/tcp            cmd
```

```
printer         515/tcp             spooler
```

Le fichier **`/etc/protocols`** contient les noms des protocoles utilisés (ip, tcp, udp, ...).

```
# Exemple de contenu du fichier /etc/protocols
# Internet (IP) protocols
#
ip 0      IP     # internet protocol, pseudo protocol number
icmp      1      ICMP   # internet control message protocol
igmp      2      IGMP   # internet group multicast protocol
ggp       3      GGP    # gateway-gateway protocol
tcp       6      TCP    # transmission control protocol
pup       12     PUP    # PARC universal packet protocol
udp       17     UDP    # user datagram protocol
```

Le fichier **`/etc/hosts.equiv`** permet de donner une liste de machines équivalentes à la machine locale.

2. Adresse d'une machine

```
struct in_addr { u_long s_addr; } ;
```

Structure `hostent`

Cette structure prédéfinie dans **`netdb.h`** correspond à une entrée dans le fichier `/etc/hosts`

```
struct {
      char *h_name;                 /* nom officiel de la machine */
      char **h_aliases;             /* liste d'alias */
      int h_addrtype;               /* type d'adresse: AF_INET, ... */
      int h_length;                 /* longueur d'adresse */
      char **h_addr_list;           /* liste d'adresses */
      #define h_addr h_addr_list[0] /* Première adresse de la liste */
                  }
```

Fonction **`bcopy`**`(b1, b2, lg)` : permet de copier une zone d'adresse b1 de longueur `lg` dans une autre zone d'adresse b2.
Fonction **`bzero`** `(adr, lg)` : permet d'initialiser à 0 une zone mémoire `adr` de taille `lg`.

Nom et adresse de la machine locale

La fonction **`gethostname`** permet de placer le nom du machine locale à l'adresse `nom`. Le paramètre `lg` indique l'espace réservé à l'adresse `nom`. La valeur de retour est 0 (si l'appel s'est bien déroulé) et -1 sinon.

```
int gethostname(nom, lg)
      char *nom;
      int lg;
```

La fonction **`gethostid`** renvoie l'adresse de la machine locale sous la forme d'un entier de quatre octets.

```
int gethostid();
```

La constante **`INADDR_ANY`** permet de laisser le système choisir une adresse à la place de l'utilisateur.

Les informations contenues dans le fichier `/etc/hosts` concernant une machine de nom ou l'adresse donné peuvent être récupérées au moyen des fonctions **`gethostbyname`** ou **`gethostbyaddr`**:

```
struct hostent *gethostbyname(nom)
                  char *nom;
struct hostent *gethostbyaddr (adresse, longueur, type)
                  char *adresse;  int longueur; int type;
```

3. Adresses de réseaux

La structure **`netent`** et les fonctions **`getnetbyname`** ou **`getnetbyaddr`** permettent d'obtenir les informations contenues dans le fichier `/etc/networks`.

```
struct netent {
      char        *n_name;          /* nom officiel du réseau */
      char        **n_aliases;      /* liste d'alias */
      int         n_addrtype;       /* type d'adresse du réseau */
      unsigned long     n_net;      /* adresse du réseau */
            }

struct netent *getnetbyname(nom)   char *nom;
struct netent *getnetbyaddr (adresse, type) char *adresse; int type;
```

4. Manipulation des ports et services

Un service sur une machine donnée se caractérise par un numéro de port du protocole correspondant. Les numéros de port jouent un rôle d'adresse pour les services. Par ailleurs, le fichier `/etc/services` contient la liste des services répertoriés. La consultation de ce fichier utilise la structure **`servent`** et les fonctions **`getservbyname`** et **`getservbyport`** :

```
struct servent {
      char  *s_name;          /* nom officiel du service */
      char  **s_aliases;      /* liste d'alias */
      int   s_port;           /* numéro de port */
      char  *s_proto;         /* protocole utilisé */
            };
struct servent *getservbyname(nom, proto)
                  char *nom; char *proto;
struct servent *getservbyport (port, proto)
            int port; char *proto;
```

5. Les protocoles

La consultation du fichier `/etc/protocols` utilise la structure **`protoent`** et les fonctions **`getprotobyname`** et **`getprotobynumber`** :

```
struct protoent {
      char  *p_name;          /* nom officiel du protocole */
      char  *p_aliases;       /* liste d'alias */
      int   p_proto;          /* numéro du protocole */
                  }

struct protoent *getprotobyname(nom) char *nom;
struct protoent *getprotobynumber (numero)int numero;
```

6. Problème de représentation des entiers

Lorsque des nombres entiers sont échangés entre machines distantes (par exemple, des adresses ou des numéros de ports), ces entiers doivent être mis préalablement sous la forme réseau. Quatre fonctions permettent de passer de la représentation locale d'un entier à sa représentation réseau ou inversement. Ces fonctions sont :

`ntohl(x)` : passage de la représentation réseau d'un entier long x à sa représentation locale.
`htonl(x)` : passage de la représentation locale d'un entier long x à sa représentation réseau.
`ntohs(x)` : passage de la représentation réseau d'un entier court x à sa représentation locale.
`htons(x)` : passage de la représentation locale d'un entier court x à sa représentation réseau.

Exemple

```
/* Programme qui permet de donner l'adresse de la machine locale et l'adresse d'une machine distante dont le
nom est saisi par l'utilisateur.  Il permet aussi de donner les numéros de port et protocole utilisés.
Ce programme est inspiré de l'exemple donné dans «  J.-M. Rifflet, "La communication sous UNIX",  page 233 »
*/

#include <stdio.h>
#include <sys/types.h>
#include <netdb.h>
#include <netinet/in.h>

main()
{
  char nom[20];
  union adr {
    int adr_i;
    unsigned char adr_c[4];
  } adr, **q;

  struct hostent *p_hote;
  struct servent *p_service;
  struct protoent *p_proto;

  gethostname(nom,20);
  adr.adr_i = htonl(gethostid());
  printf("\n Machine locale: %s\n",nom);
  printf("\n Adresse locale: %u.%u.%u.%u\n",adr.adr_c[0],adr.adr_c[1],adr.adr_c[2],adr.adr_c[3]);
  printf("\n Nom de machine distante ? : ");
  gets(nom);

  if((p_hote = gethostbyname(nom)) == NULL )
    printf("\n Machine %s inconnue\n",nom);
 else {
  q= (union adr **)p_hote->h_addr_list;
  printf("\n Adresse distante : %u.%u.%u.%u\n",
    (*q)->adr_c[0],(*q)->adr_c[1],(*q)->adr_c[2],(*q)->adr_c[3]);
      }
  /* Numero du protocole tcp */
  p_proto = getprotobyname("tcp");
  printf("\n Protocole tcp: %d\n",p_proto->p_proto);
  /* accès aux caractéristiques du protocole ftp */
  p_service = getservbyname("ftp","tcp");
  printf("\n Service ftp => port %d \n",p_service->s_port);
  printf("\n Protocole: %s\n",p_service->s_proto);
}
```