

Utiliser les « sockets » UNIX

Présentation

Les *sockets* sont apparus sur les versions BSD d'UNIX. Comme les tubes elles permettent de faciliter la communication de données entre deux ou plusieurs processus. Mais contrairement aux tubes, la communication peut s'effectuer dans les deux sens et ces processus peuvent s'exécuter sur une ou plusieurs machines distantes. Dans ce dernier cas, le réseau est mis à contribution pour transférer les flots de données. La création d'une socket passe par la fonction :

```
int socket ( int domaine, int type, int protocole );
```

Le *domaine* peut-être AF_UNIX (pour *Adress Family UNIX*), dans ce cas la socket ne permet que la communication entre processus d'une même machine ; ou AF_INET si les communications passent par le réseau. Parmi les quatre types possibles, le *type* SOCK_DGRAM (pour socket de type datagramme) correspond à une communication par envoi de messages (2ko maximum par message). Le *protocole* 0 est le protocole par défaut. Cette fonction renvoie -1 en cas d'erreur.

Attachement d'une socket

Pour que deux processus *p1* et *p2* (s'exécutant respectivement sur les machines *m1* et *m2*) communiquent au travers de deux sockets de type réseau (*s1* sur *m1* et *s2* sur *m2*) il faut que *p1* « connaisse » *s2* et que *p2* « connaisse » *s1*. Cette reconnaissance passe par l'existence de *boîtes à lettre* appelées des *ports*. Chaque port est repéré par un numéro. A chaque socket est donc rattaché un numéro de port qui permet de « parler » de cette socket sur la machine distante. Le rattachement d'une socket passe par la procédure

```
int bind ( int socket, struct sockaddr* adr, int lenadr);
```

Si (par exemple) le processus *p2* désire écouter les messages qui arrivent sur le port *n* de la machine *m2* il faut que *p2* réalise l'attachement de *s2* par un appel à la fonction `bind`. Pour ce faire, il doit préparer une structure `sockaddr_in` dans laquelle le champ `sin_addr` est initialisé à `INADDR_ANY` (c'est un moyen simple pour parler de sa propre adresse).

Dernière remarque : l'utilisation en émission d'une socket qui n'a pas été attachée à un port entraîne automatiquement sont attachement. Dans ce cas, le choix du port est laissé à la discrétion du système.

Envoyer ou recevoir un message

L'envoi d'un message (c'est la seule option possible pour les sockets de type datagramme) est réalisé par un appel à la fonction

```
int sendto(  
    int socket,                /* descripteur de la socket d'emission */  
    char* msg,                 /* adresse du message a envoyer      */  
    int lg,                    /* longueur du message                */  
    int option,                /* toujours 0 pour le type SOCK_DGRAM */  
    struct sockaddr* dest;     /* l'adresse de la socket destination */  
    int lgdest                 /* longueur de cette adresse         */  
);
```

La réception d'un message (ou l'attente si aucun message n'est arrivé) est réalisée par un appel à la fonction

```
int recvfrom(
    int socket,           /* descripteur de la socket de reception */
    char* msg,           /* adresse de recuperation du message */
    int lg,              /* taille de l'espace alloue a msg */
    int option,          /* toujours 0 pour le type SOCK_DGRAM */
    struct sockaddr* dest; /* pour recuperer l'adr de l'expediteur */
    int* lgdest          /* taille espace reserve a dest */
);
```

Définition des structures utiles

La manipulation des adresses (de machines sur le réseau) pose quelques problèmes avec les sockets. En effet, les sockets peuvent être locales (à la machine) ou globales (au réseau). Les adresses sont donc de type UNIX dans le premier cas (c'est la structure `sockaddr_un`) ou de type réseau dans le second (c'est la structure `sockaddr_in` qui est définie ci-dessous).

```
struct sockaddr_in {
    short      sin_family;   /* famille: AF_INET */
    u_short    sin_port;    /* le numero de port */
    struct in_addr sin_addr; /* l'adresse internet */
    char       sin_zero[8]; /* un champ de 8 zeros */
};

struct in_addr { unsigned long s_addr; };
```

D'un autre coté, les fonctions de manipulation des sockets réclament des adresses au format `sockaddr`. Nous sommes donc contraints d'utiliser des changements de type pour transformer un pointeur vers une structure `sockaddr_in` en un pointeur vers une structure `sockaddr`. D'ailleurs ce cas de figure est prévu et ces mêmes fonctions réclament également la taille de ces adresses.

Conversion des entiers

Pour se prémunir des variations de codage pouvant apparaître sur des machines différentes, les entiers utilisés (notamment les numéros de ports) doivent être codés avant de passer sur le réseau pour être décodés après leur récupération. Pour cette étape de codage/décodage vous devez utiliser les fonctions suivantes :

- `htons()` : *Host short TO Network Short*
- `ntohs()` : *Network short TO Host Short*