Les structures de données IP en C

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 5 mars 2009. Dernière mise à jour le 6 mars 2009

https://www.bortzmeyer.org/ip-data-structures.html

Comme je ne programme pas en C tous les jours, à chaque fois que je m'y remets (et c'est en général pour un programme réseau), j'ai du mal avec les structures de données qui servent à IP. Alors, pour garder trace de ce que j'ai compris la dernière fois, le plus simple est encore d'en faire profiter l'univers en mettant quelques notes sur ce blog.

Les structures de données qui stockent les adresses IP, les métadonnées sur ces adresses comme leur famille (IPv4 ou IPv6), les prises, etc, sont complexes. Mais c'est le résultat d'un choix de faire assez général, notamment pour permettre au même programme de gérer IPv4 et IPv6, avec le minimum de modifications.

Donc, la structure de données la plus importante est la `struct addrinfo`. Elle stocke tout ce qu'il faut pour se connecter à une machine, notamment son adresse IP. Pour pouvoir être chaînée à d'autres `struct addrinfo`, elle se termine par un pointeur vers la structure suivante. Sa définition officielle est dans le RFC 3493 [1] et on trouve le code (un peu plus complexe qu'ici) dans `/usr/include/netdb.h` :

```
struct addrinfo {
        int             ai_flags;       /* AI_PASSIVE, AI_CANONNAME */
        int             ai_family;      /* PF_xxx */
        int             ai_socktype;    /* SOCK_xxx */
        int             ai_protocol;    /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
        socklen_t        ai_addrlen;    /* length of ai_addr */
        char            *ai_canonname;  /* canonical name for hostname */
        struct sockaddr *ai_addr;       /* binary address */
        struct addrinfo *ai_next;       /* next structure in linked list */
};
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc3493.txt`

`ai_family` identifie la **famille**, IPv4 ou IPv6. C'est le champ `ai_addr` qui stocke l'adresse ou plutôt les adresses (c'est une liste chaînée).

On voit donc que les adresses sont de type `struct sockaddr`. Ce type est défini dans `/usr/include/sys/soc` (parfois via une inclusion d'un autre fichier) et vaut :

```
struct sockaddr {
        uint8_t       sa_len;         /* total length */
        sa_family_t     sa_family;      /* address family */
        char            sa_data[14];    /* address value */
};
```

Le point important est que c'est une structure générique, qui marche aussi bien pour IPv4 que pour IPv6. Pour extraire des informations pertinentes à ces deux protocoles, il faut convertir les `struct sockaddr` génériques en `struct sockaddr_in` et `struct sockaddr_in6` spécifiques. Cela se fait en général en convertissant les pointeurs (exemple ci-dessous lors de l'utilisation de `inet_ntop()`).. La structure pour IPv6 vaut (`/usr/include/netinet/in.h`) :

```
struct sockaddr_in6 {
    uint8_t         sin6_len;       /* length of this struct */
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces for a scope */
};
```

et celle pour IPv4 :

```
struct sockaddr_in {
        uint8_t         sin_len;
        sa_family_t     sin_family;
        in_port_t       sin_port;
        struct in_addr  sin_addr;
        __int8_t        sin_zero[8];
};
```

Notez la "*padding*" de huit octets à la fin, qui lui permet d'avoir exactement la même taille que son équivalente IPv6. Normalement, ce n'est pas garanti, si on veut une structure de données qui puisse réellement contenir les deux familles d'adresses, on devrait utiliser `struct sockaddr_storage` (RFC 3493, section 3.10).

Nous nous approchons désormais sérieusement des adresses IP tout court. Il reste une étape, regarder le champ `sin_addr` ou `sin6_addr`. Il est de type `struct in_addr` ou `struct in6_addr`. La définition de ces structures est également dans `/usr/include/netinet/in.h` (`in6_addr` est présentée à la section 3.2 du RFC 3493) :

```
struct in_addr {
    uint32_t s_addr;
  };
...
struct in6_addr {
          uint8_t  s6_addr[16];
};
```

Les deux structures stockent la représentation binaire des adresses IP, celle qui circule sur le câble : quatre octets pour IPv4 et 16 pour IPv6. Si vous regardez le fichier `.h`, vous verrez que `in6_addr` est souvent représentée sous forme d'une union, pour faciliter l'alignement.

Maintenant qu'on a ces structures de données, comment les utilise t-on ? Pour remplir les `struct addrinfo`, on utilise `getaddrinfo()` qui a remplacé `gethostbyname()` il y a de très nombreuses années. Par exemple, si le nom de la machine qu'on cherche est dans `server` (un simple `char *`) :

```
/* hints and res are "struct addrinfo", the first one is an IN
parameter, the second an OUT one. */
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC; /* v4 or v6, I don't care */
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(server, port_name, &hints, &res);
/* Now, "res" contains the IP addresses of "server" */
/* We can now do other things such as creating a socket: */
sockfd = socket(res->ai_family, res->ai_socktype, protocol);
/* And connecting to it, since connect() takes a "struct sockaddr" as input: */
connect(sockfd, res->ai_addr, res->ai_addrlen);
```

Notez que le code ci-dessus ne parcourt par la liste des adresses renvoyée par `getaddrinfo()`, seule la première est utilisée (`ai_next` est ignorée).

Dans la `struct addrinfo` passée en premier paramètre à `getaddrinfo()`, une option est particulièrement intéressante, `AI_NUMERICHOST` qui permet de s'assurer qu'une chaîne de caractères, donnée par l'utilisateur, a bien la syntaxe d'une adresse IP :

```
hints_numeric.ai_flags = AI_NUMERICHOST;
error = getaddrinfo(server, port_name, &hints_numeric, &res);
if (error && error == EAI_NONAME) { /* It may be a name, but
it is certainly not an address */
... /* Handle it as a name */
```

Et pour afficher des adresses à l'utilisateur humain ? Les structures `in_addr` et `in6_addr` stockent la forme binaire de l'adresse mais, ici, on veut la forme texte <https://www.bortzmeyer.org/representation-texte.html>. Pour cela, deux fonctions sont fournies, `inet_ntop()` pour traduire l'adresse en texte et `inet_pton()` pour le contraire. Supposons que la variable `myaddrinfo` soit une `struct addrinfo`, afficher la première adresse IP nécessite d'extraire `ai_addr`, de le convertir dans la `struct sockaddr` du bon type (`struct sockaddr_in6` pour IPv6) et d'appeler `inet_ntop()` avec les bons paramètres, qui dépendent de la famille :

```
struct addrinfo *myaddrinfo;
struct sockaddr *addr;
struct sockaddr_in6 *addr6;
addr = myaddrinfo->ai_addr;
char *address_text;
...
if (myaddrinfo->ai_family == AF_INET6) {
       address_text = malloc(INET6_ADDRSTRLEN);
       addr = myaddrinfo->ai_addr;
       addr6 = (struct sockaddr_in6 *) addr; /* Conversion between the
                     various  struct sockaddr* is always possible,
      since they have the same layout */
       inet_ntop (AF_INET6, &addr6->sin6_addr, address_text,
                        INET6_ADDRSTRLEN);
}
else /* IPv4... */
```

Un descripteur de prise, lui, est très simple, c'est un bête entier :

```
int             sockfd;
...
sockfd = socket(...);
```

Il pointe, via la table des descripteurs (que vous ne manipulez jamais) vers la vraie prise. C'est toujours le descripteur qu'on passera comme paramètre pour agir sur la prise, par exemple pour la lier à une adresse donnée :

```
bind(sockfd, ...);
```

L'article "*IP Addresses, structs, and Data Munging*" <http://beej.us/guide/bgnet/output/html/multipage/ipstructsdata.html> m'a beaucoup aidé et fournit une excellente introduction. Pour tout connaître en détail, la référence est bien sûr le livre de Stevens, "*Unix Network Programming*" <https://www.bortzmeyer.org/unix-network-programming.html>. Merci à Samuel Tardieu pour ses remarques sur la conversion des `struct sockaddr`.