TP n°9

TP à brac

1 Retour sur DNS

Nous avons déjà vu, lors du TP 4, comment faire le résolution de noms d'hôte, avec l'ancien API de `gethostbyname(3)` et le nouveau de `getaddrinfo(3)`. On va voir un peu plus en détail comment ça marche.

1.1 Les entrailles de DNS

Commencez par faire un `host -a` sur une adresse (`univ-paris-diderot.fr` par exemple). Vous allez recevoir plein d'informations ; essayez (si besoin à l'aide de la page man de `host(1)`) de voir ce qu'ils signifient.

Vous pouvez interroger un serveur spécifique en ajoutant son adresse à votre commande `host` : par exemple `host www.lemonde.fr truc` demande à un serveur DNS `truc` de nous donner son opinion sur l'adresse IP de `www.lemonde.fr`. Depuis les salles de TP, cela n'est pas possible mais vous pouvez essayer de faire les points suivants depuis chez vous :

- Qu'est-ce qui se passe quand vous faites
  `host -a www.lemonde.fr a.root-servers.net` ?
- Est-ce que vous avez assez d'informations pour savoir l'adresse IP du Monde ? Et si non, comment pourriez-vous avoir les informations qu'il faut ?

1.2 `getaddrinfo` revisité

Dans le TP 4 (suite), on vous a déjà sommairement présenté l'API `getaddrinfo(3)` ; on vous rappelle le fonctionnement basique :

La fonction `getaddrinfo(3)` prend en argument un nom d'hôte, un nom de service, un pointeur sur une structure de *hints*, et un pointeur sur un résultat. Elle renvoie un entier qui sera 0 en cas de réussite, ou une valeur d'erreur qui peut être interpretée avec `gai_strerror(3)`.

Il va utiliser le nom d'hôte, le nom de service et les *hints* pour vous donner une liste chaînée de structures `addrinfo` (décrites dans le TP 4 (suite)).

Les *hints* prennent aussi le forme d'une structure `addrinfo` : vous pouvez indiquer des restrictions sur le type de socket que vous souhaitez utiliser (si vous n'avez aucune préférence, vous mettez `NULL`).

Dans votre structure de hints, les champs suivants peuvent avour une valeur utile :

`ai_family` La famille de protocoles à utiliser ; par exemple `PF_INET` pour IPv4, `PF_INET6` pour IPv6 ou bien `PF_UNSPEC` pour n'importe quelle famille.

`ai_socktype` Le type de socket : `SOCK_STREAM`, `SOCK_DGRAM` ou `SOCK_RAW`. Si vous prenez tous les types, mettez zéro.

`ai_protocol` Le protocole : `IPPROTO_UDP` ou `IPPROTO_TCP`, ou encore zéro pour n'importe lequel.

`ai_flags` Il y en a cinq. Vous pouvez combinez les flags en faisant un or bit a bit.

- `AI_ADDRCONFIG` Indique qu'une adresse IPv4 doit être retournée seulement si votre ordinateur a une adresse IPv4 configurée, et de même pour IPv6.
- `AI_CANONNAME` Indique que une chaîne de caractères avec le nom canonique du nom d'hôte doit être mis dans le champ `ai_canonname` du premier élément `addrinfo` retourné.
- `AI_NUMERICHOST` Indique que le nom d'hôte est en forme numérique, c'est à dire du forme `192.168.0.1` ; il n'y aura pas de résolution de nom.
- `AI_NUMERICSERV` Indique que le nom de service est en forme numérique. Il n'y aura pas de résolution de nom (par exemple par travers NIS ou le fichier `/etc/services`).
- `AI_PASSIVE` Indique que le socket sera utilisée pour un `bind(2)` ; dans ce cas, si le nom d'hôte est `NULL`, le socket sera connecté à tous les adresses configurés de votre ordinateur.

Les autres éléments de la structure `addrinfo` doivent être zéro ou `NULL`, selon leur type.

Après appèl (avec succès) de `getaddrinfo(3)`, vous aurez une liste chaînée de structures `addrinfo`, dont vous pouvez utiliser les champs `ai_family`, `ai_socktype` et `ai_protocol` pour un appel à `socket(2)`. Vous pouvez également utiliser `ai_addr` et `ai_addrlen` pour des appels à `bind(2)` ou `connect(2)`.

2 Programmation en double stack

On va utiliser `getaddrinfo` pour écrire une application assez simple. Il y a des sites web (`www.kame.net`, par exemple) qui donnent une réponse différente selon si on se connecte en IPv4 ou en IPv6. Ecrivez un programme qui peut détecter cette circonstance.

3 Comment ça se passe en vrai

Pour résoudre des problèmes réseau, faire du reverse engineering, ou des choses dans ce genre, il est parfois intéressant de pouvoir regarder ce qui se passe sur le fil. Pour cela, nous avons le programme `tcpdump(1)` (disponible dans `/usr/sbin` sur `lucien`).

Un exemple de ce qu'on peut voir :

```
09 :27 :59.097053 IP 192.168.0.22.42387 > 145.58.28.92.80 : Flags [S],
seq 4198830417, win 65535, options [mss 1460,nop,wscale 3,sackOK,TS val
1047259 ecr 0], length 0
09 :27 :59.138396 IP 145.58.28.92.80 > 192.168.0.22.42387 : Flags [S.],
seq 1459724888, ack 4198830418, win 5792, options [mss 1460,sackOK,TS val
215744361 ecr 1047259,nop,wscale 7], length 0
09 :27 :59.138434 IP 192.168.0.22.42387 > 145.58.28.92.80 : Flags [.], ack
1, win 8326, options [nop,nop,TS val 1047301 ecr 215744361], length 0
```

Vous voyez ici le début d'une connection HTTP (les trois premiers paquets). Par ligne, on peut voir :

- le temps ;
- les adresses source (IP et port) et destination ;
- les flags TCP (S pour SYN et . pour ACK ; il y en a d'autres que vous pouvez trouver dans la page man de `tcpdump(1)`) ;
- le nombre de séquence (`seq`) et/ou d'accusé de réception (`ack`) ;
- la taille de la fenêtre de réception (`win`) ;
- des options éventuels ;
- et finalement la taille du paquet (`length`).

Sur `http://www.pps.jussieu.fr/~boender/data.bin`, vous trouverez un fichier dans lequel j'ai capté quelque temps de trafic IP sur mon ordinateur ; vous pouvez le lire avec l'option `-r` de tcpdump.

La question est simple : il y a au moins trois choses que j'ai faites pendant que j'ai capté ce fichier. Est-ce que vous pouvez trouver lesquels ?

Un conseil : regardez bien la page man de `tcpdump`, il y a plein d'options intéressantes. Par exemple, vous pouvez filtrer des paquets qui ne vous intéressent pas.