ASR.3.2 TP #N

Applications Client/Serveur
IPv4-versus-IPv6 indépendantes

Type non connecté

I. Obtention d'informations sur les adresses sockets valides pour un host.

Les fonctions `getaddrinfo` et `getnameinfo`

Ecrire un programme `get-host-info.c` qui prend un nom/adresse de host en premier argument, un numéro/nom de service en deuxième argument et affiche toutes les adresses de sockets valides pour ces deux paramètres. On rappelle que `getaddrinfo` n'a pas vocation à vérifier que le service ou le numéro de port sont ouverts dans la machine dont le nom ou l'adresse ip lui sont donnés en argument.

Pour les correspondances entre les noms et les numéros, on peut se référer à :

1. `/usr/include/bits/socket.h` pour les familles d'adresses ;
2. `/usr/include/bits/socket_type.h` pour les types de sockets ;
3. `/etc/protocols` pour les protocoles.

```
selma@salle225-05:~ $ ./get-host-info "www.google.fr" "http"
172.217.18.195.http
fam: 2 -- socktype: 1 -- pro: 6 -- addrlen: 16 -- canonname: www.google.fr
172.217.18.195.http
fam: 2 -- socktype: 2 -- pro: 17 -- addrlen: 16 -- canonname: (null)
172.217.18.195.http
fam: 2 -- socktype: 1 -- pro: 132 -- addrlen: 16 -- canonname: (null)
172.217.18.195.http
fam: 2 -- socktype: 5 -- pro: 132 -- addrlen: 16 -- canonname: (null)
2a00:1450:4007:805::2003.http
fam: 10 -- socktype: 1 -- pro: 6 -- addrlen: 28 -- canonname: (null)
2a00:1450:4007:805::2003.http
fam: 10 -- socktype: 2 -- pro: 17 -- addrlen: 28 -- canonname: (null)
2a00:1450:4007:805::2003.http
fam: 10 -- socktype: 1 -- pro: 132 -- addrlen: 28 -- canonname: (null)
2a00:1450:4007:805::2003.http
fam: 10 -- socktype: 5 -- pro: 132 -- addrlen: 28 -- canonname: (null)

selma@salle234-01:~ $ ./get-host-info fe80::6600:6aff:fe8d:1a2b%eno1 "ssh"
fe80::6600:6aff:fe8d:1a2b%eno1.ssh
fam: 10 -- socktype: 1 -- pro: 6 -- addrlen: 28 -- canonname: fe80::6600:6aff:fe8d:1a2b%eno1
fe80::6600:6aff:fe8d:1a2b%eno1.ssh
fam: 10 -- socktype: 2 -- pro: 17 -- addrlen: 28 -- canonname: (null)
fe80::6600:6aff:fe8d:1a2b%eno1.ssh
fam: 10 -- socktype: 1 -- pro: 132 -- addrlen: 28 -- canonname: (null)
fe80::6600:6aff:fe8d:1a2b%eno1.ssh
fam: 10 -- socktype: 5 -- pro: 132 -- addrlen: 28 -- canonname: (null)

selma@salle225-05:~ $ ip a
eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 64:00:6a:8d:1a:2b brd ff:ff:ff:ff:ff:ff
    inet 172.16.2.78/16 brd 172.16.255.255 scope global noprefixroute eno1
       valid_lft forever preferred_lft forever
    inet6 fe80::6600:6aff:fe8d:1a2b/64 scope link
.....
......
```

```
selma@salle225-05:~ $ ./get-host-info "www.archlinux.org" "ftp"
138.201.81.199.ftp
fam: 2 -- socktype: 1 -- pro: 6 -- addrlen: 16 -- canonname: apollo.archlinux.org
138.201.81.199.ftp
fam: 2 -- socktype: 2 -- pro: 17 -- addrlen: 16 -- canonname: (null)
138.201.81.199.ftp
fam: 2 -- socktype: 1 -- pro: 132 -- addrlen: 16 -- canonname: (null)
138.201.81.199.ftp
fam: 2 -- socktype: 5 -- pro: 132 -- addrlen: 16 -- canonname: (null)
2a01:4f8:172:1d86::1.ftp
fam: 10 -- socktype: 1 -- pro: 6 -- addrlen: 28 -- canonname: (null)
2a01:4f8:172:1d86::1.ftp
fam: 10 -- socktype: 2 -- pro: 17 -- addrlen: 28 -- canonname: (null)
2a01:4f8:172:1d86::1.ftp
fam: 10 -- socktype: 1 -- pro: 132 -- addrlen: 28 -- canonname: (null)
2a01:4f8:172:1d86::1.ftp
fam: 10 -- socktype: 5 -- pro: 132 -- addrlen: 28 -- canonname: (null)
```

II. Un client/Serveur de type non connecté avec IPv4/6 indépendance.

Objectifs. Ecrire des programme `udp-unspec-srv.c` et `udp-unspec-clt.c` qui sont respectivement un serveur udp et un client udp utilisant des familles d'adresses unspécifiées (sans dépendance IPv4 ou IPv6).

Spécifications.

- Le serveur attache un numéro de port à une socket locale puis, à travers une boucle infinie réceptionne (appel système `recvfrom`) des datagrammes UDP sur cette socket, et affiche ce qu'il reçoit précédé de `peer_addr_ip.peer_port`, où `peer_addr_ip` est l'adresse du client dont on a reçu le message et `peer_port` est le numéro de port de ce client. L'affichage est suivi sur la même ligne par le nombre d'octets reçus. Le programme `udp-unspec-srv.c` utilise un paramètre à la ligne de commande : le numéro de port.
- Le client envoie (appel système `sendto`) une suite de messages composés par l'utilisateur à un serveur udp. Il utilise deux paramètres à la ligne de commandes : **le nom du serveur** et le numéro de port du service. Les messages sont saisis à travers une boucle sur `read(0,..,..)`. L'utilisateur décide d'arrêter par l'envoi de `CTRL-D` (rappel : signifie fin de fichier sur l'entrée standard).
- Le type `struct sockaddr_storage` pour les adresses de sockets dispense de devoir savoir si c'est un `struct sockaddr_in` ou un `struct sockaddr_in6`. Il a une taille assez large pour contenir tout type d'adresse de socket.
- Dans la structure `struct addrinfo` utilisée pour indiquer à `getaddrinfo` les critères sur les adresses de sockets souhaitées, serveur et client indiquent `AF_UNSPEC` dans le champ `ai_family`.

Mise en œuvre.

1. Côté serveur.
   (a) Après récupération de la liste retournée par `getaddrinfo`, le serveur appelle la fonction :

   ```
   int sock_bind(struct addrinfo *res,struct addrinfo *s);
   ```

   Dans cette fonction, il teste chaque cellule de la liste chaînée `res` jusqu'à ce que les appels systèmes `socket` et `bind` réussissent simultanément. La fonction copie alors dans `*s` la structure `struct addrinfo` correspondant à la première cellule de `res` pour laquelle `socket` et `bind` ont simultanément réussi. La fonction retourne le descripteur de socket obtenu.

   (b) Au retour de la fonction `sock_bind`, on veut se dépêcher de libérer la mémoire utilisée pour les besoins de la liste chaînée. Comme le champ `ai_addr` de la structure `struct addrinfo` est un pointeur vers l'adresse de socket (stockée dans le heap), il faudra, au préalable, copier cette adresse de socket dans un objet, disons, `scksrv` de type `struct sockaddr_storage`.

   (c) On libère alors la mémoire utilisée par la liste retournée par `getaddrinfo`.

   (d) Le serveur appelle ensuite la fonction :

   ```
   int print_where_bound(struct sockaddr *sck, struct addrinfo s);
   ```

   à l'aide de laquelle on affiche les coordonnées où la socket a été attachée. La fonction retourne `0`/`-1` selon que l'appel à la fonction `getnameinfo` qui y a été fait ait réussi ou pas.

(e) Lancer le programme et vérifier à l'aide de la commande `ss` que le port est bien ouvert sur la machine.

(f) Tester le programme à l'aide de plusieurs exemplaires simultanés de `ncat` en guise de clients.

```
selma@salle225-05:~ $ ./udp-unspec-srv 49500
Successfully bound to 0.0.0.0.49500
fam: 2 -- socktype: 2 -- pro: 17 -- addrlen: 16
```

Dans un autre terminal, on vérifie qu'on a bien attaché au numéro de port passé au programme :

```
selma@salle225-05:~ $ ss -na --udp
State    Recv-Q    Send-Q         Local Address:Port        Peer Address:Port
.............................
.............................
UNCONN   0         0                  0.0.0.0:49500             0.0.0.0:*
```

(g) Le serveur entre ensuite dans une boucle infinie de réceptions/affichages des datagrammes reçus avec affichage des coordonnées des clients correspondants. Il aura enregistré au préalable ces coordonnées dans une `struct sockaddr_storage`. Il passe ces coordonnées à la fonction `getnameinfo` qui les lui formate sous forme chaînes de caractères.

```
From salle229-02.arda.60964 --> 12345678 -- Total: 8 bytes
From salle229-02.arda.60964 --> 1234567 897654321 -- Total: 17 bytes
From salle234-01.arda.41290 --> ABCDEFGHIJK LMNOP -- Total: 17 bytes
From salle234-01.arda.41290 --> ZYXWV UTS RQP -- Total: 13 bytes
From salle229-02.arda.60964 --> 11111 22222 3333 -- Total: 16 bytes
```

2. Côté client.

(a) Après récupération de la liste retournée par `getaddrinfo`, le client appelle la fonction :

```
int try_socket(struct addrinfo *res,struct addrinfo *s);
```

Dans cette fonction, il teste chaque cellule de la liste chaînée `res` jusqu'à ce que l'appel système `socket` réussisse. La fonction copie alors dans `*s` la structure `struct addrinfo` correspondant à la première cellule pour laquelle `socket` a réussi. La fonction retourne le descripteur de socket obtenu.

(b) Dans un objet, disons, `scksrv` de type `struct sockaddr_storage`, on retient alors les coordonnées du serveur et on libère la mémoire utilisée par la liste chaînée retournée par `getaddrinfo`.

(c) Le client affiche les coordonnées du serveur auquel il s'apprête à envoyer des datagrammes.
Une fois qu'on a des programmes qui tournent correctement, on lancera le serveur sur une machine distante et on lancera plusieurs clients simultanés.

```
selma@salle229-02:~ $ ./udp-unspec-clt salle225-05.arda 49500
Ready to send datagrams to server 172.16.2.78.49500...
```

(d) Il entre ensuite dans une boucle de saisie de messages de la part de l'utilisateur en utilisant `read(0,...,...)`. Chaque message saisi est envoyé au serveur. L'utilisateur décide d'en finir par `CTRL-D`.

```
Your message: 12345678
Your message: 1234567 897654321
Your message: 11111 22222 3333
Your message:
```

```
selma@salle234-01:~ $ ./udp-unspec-clt salle225-05.arda 49500
Ready to send datagrams to server 172.16.2.78.49500...
Your message: ABCDEFGHIJK LMNOP
Your message: ZYXWV UTS RQP
Your message:
```