ASR.3.2 TP #N+1

Applications Client/Serveur
IPv4-versus-IPv6 indépendantes

Type connecté

Schéma multiprocessus

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

Spécifications.

- Le serveur attache un numéro de port à une socket locale. Pour tester serveur et client sur une même machine, ne pas positionner `AI_PASSIVE` dans le champ `ai_flags` du troisième argument de `getaddrinfo`. Le serveur se met ensuite sur une boucle infinie dans laquelle, il attend des demandes de connexions (appel système `accept`). A l'arrivée d'une telle demande, il crée un processus fils qui doit se charger du service avec le client. Ici, ce service consistera simplement à renvoyer au client le message qu'il envoie au serveur. Le processus fils est dans une boucle infinie où il réceptionne (appel système `recv`) les messages du client et les lui renvoie aussitôt (appel système `send`). Voir plus loin comment les affichages doivent être réalisés.
- Le client envoie au serveur (appel système `send`) une suite de messages composés par l'utilisateur. 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).
- 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 `res` 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 système `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. On libère alors la mémoire utilisée par la liste retournée par `getaddrinfo`.
   (b) Le serveur marque le descripteur de socket obtenu comme acceptant les demandes de connexions (appel système `listen`).
   (c) Le serveur appelle ensuite la fonction :

   ```
   int print_where_bound(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.
   (d) Il entre ensuite dans une boucle infinie ou il réalise `accept, fork`.

(e) Chaque fils devra fermer le descripteur de socket d'écoute hérité de son parent, puis appeler la fonction `void serv_client`. Le premier argument de cette fonction est de type `struct sockaddr *` par lequel on passe à la fonction l'adresse de socket du client que le parent a enregistré dans le deuxième argument de `accept`. Le deuxième argument de `void serv_client` est la longueur de l'adresse de socket du client que le parent a enregistré dans le troisième argument de `accept`. Dans la fonction, le processus fils réalise des `recv`, `send`, dans une boucle infinie.

(f) Parent et enfants, devront gérer le signal `SGINT` pour fermer correctement les descripteurs de socket avant de se terminer. Le parent devra gérer le signal `SIGCHLD` pour ne pas laisser subsister ses enfants zombies.

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

```
selma@salle231-01 $ ncat localhost 49500
1111111111111111111
1111111111111111111
2222222
2222222


333333333333333333
333333333333333333 //Après ceci, je vais donner CTRL-D. Le serveur détecte cette fin.
selma@salle231-01 $
```

Pendant que le précédent `ncat` tourne, un autre `ncat` tourne dans un autre terminal :

```
selma@salle231-01 $ ncat localhost 49500
AAAAAA
AAAAAA
BBBBBBBBBBBBBB
BBBBBBBBBBBBBB
```

Au même moment côté serveur :

```
selma@salle231-01 $ ./a.out 49500
Successfully bound to ::1.49500 -- Listening...
       fam: 10 -- socktype: 1 -- pro: 6 -- addrlen: 28
CHLD 17159: Starting service with client localhost.45146 .....
From localhost.45146 --> 1111111111111111111
  Total: 20 bytes
From localhost.45146 --> 2222222
  Total: 8 bytes
From localhost.45146 -->
  Total: 1 bytes
CHLD 17177: Starting service with client localhost.45150 .....
From localhost.45150 --> AAAAAA
  Total: 7 bytes
From localhost.45146 --> 333333333333333333
  Total: 19 bytes
From localhost.45150 --> BBBBBBBBBBBBBB
  Total: 15 bytes
CHLD 17159: Client localhost.45146 has left. Closing socket descriptor 4 and exiting....
From localhost.45150  ....
......
.....
^C
CHLD PROCESS 17177 closing socket desciptor 4 and exiting...

Main process 17154 closing socket descriptor 3 and exiting...
```

2. Côté client.

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

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

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

   (b) Le client affiche les coordonnées du serveur auquel il s'apprête à envoyer des segments.

   (c) 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`.

Une fois qu'on a des programmes qui tournent correctement, on lancera serveur et clients sur des machines distantes : `AI_PASSIVE` devra alors être positionné (côté serveur).

Je lance un premier client que j'arrêterai pas `CTRL-D`, puis je relancerai un autre dans le même terminal. Pendant ce temps, un autre client est lancé sur une autre machine. J'enverrai ensuite `CTRL-C` au serveur.

```
selma@salle229-05 $ ./clt salle231-01.arda 49500
Connexion established with server 172.16.2.107.49500...
Your message: 1111111111111
From Server 172.16.2.107.49500 --> 1111111111111
  Total: 14 bytes
Your message: 222222222222222222
From Server 172.16.2.107.49500 --> 222222222222222222
  Total: 22 bytes
Your message: 3333333333333
From Server 172.16.2.107.49500 --> 3333333333333
  Total: 14 bytes
Your message: 44444444444444444444
From Server 172.16.2.107.49500 --> 44444444444444444444
  Total: 22 bytes
Your message: 555555555555555
From Server 172.16.2.107.49500 --> 555555555555555
  Total: 16 bytes
Your message:  selma@salle229-05 $ ./clt salle231-01.arda 49500
Connexion established with server 172.16.2.107.49500...
Your message: 66666666666666
From Server 172.16.2.107.49500 --> 66666666666666
  Total: 15 bytes
Your message: 77777777777
From Server 172.16.2.107.49500 --> 77777777777
  Total: 12 bytes
Your message:
Server 172.16.2.107.49500 has closed.
selma@salle229-05 $

selma@salle235-10 $ ./clt salle231-01.arda 49500
Connexion established with server 172.16.2.107.49500...
Your message: AAAAAAAAAA
From Server 172.16.2.107.49500 --> AAAAAAAAAA
  Total: 11 bytes
Your message: BBBBBBB
From Server 172.16.2.107.49500 --> BBBBBBB
```

```
  Total: 8 bytes
Your message: CCCCCCCCCCCCCCC
From Server 172.16.2.107.49500 --> CCCCCCCCCCCCCCC
  Total: 17 bytes
Your message: DDDDDDDDDDDDD
From Server 172.16.2.107.49500 --> DDDDDDDDDDDDD
  Total: 15 bytes
Your message: EEEEEEEEEEE
From Server 172.16.2.107.49500 --> EEEEEEEEEEE
  Total: 13 bytes
Your message: FF
From Server 172.16.2.107.49500 --> FF
  Total: 3 bytes
Your message:
Server 172.16.2.107.49500 has closed.
selma@salle235-10 $
```

Pendant ce temps, côté serveur :

```
selma@salle231-01 $ ./a.out 49500
Successfully bound to 0.0.0.0.49500. Listening...
fam: 2 -- socktype: 1 -- pro: 6 -- addrlen: 16
CHLD 23742: Starting service with client salle229-05.arda.48068 .....
From salle229-05.arda.48068 --> 1111111111111
  Total: 14 bytes
From salle229-05.arda.48068 --> 222222222222222222
  Total: 22 bytes
From salle229-05.arda.48068 --> 3333333333333
  Total: 14 bytes
CHLD 23752: Starting service with client salle235-10.arda.43026 .....
From salle235-10.arda.43026 --> AAAAAAAAAA
  Total: 11 bytes
From salle235-10.arda.43026 --> BBBBBBB
  Total: 8 bytes
From salle229-05.arda.48068 --> 44444444444444444444
  Total: 22 bytes
From salle235-10.arda.43026 --> CCCCCCCCCCCCCCC
  Total: 17 bytes
From salle235-10.arda.43026 --> DDDDDDDDDDDDD
  Total: 15 bytes
From salle229-05.arda.48068 --> 555555555555555
  Total: 16 bytes
CHLD 23742: Client salle229-05.arda.48068 has left. Closing socket descriptor 4 and exiting....
CHLD 23782: Starting service with client salle229-05.arda.48072 .....
From salle229-05.arda.48072 --> 66666666666666
  Total: 15 bytes
From salle229-05.arda.48072 --> 77777777777
  Total: 12 bytes
From salle235-10.arda.43026 --> EEEEEEEEEEE
  Total: 13 bytes
From salle235-10.arda.43026 --> FF
  Total: 3 bytes
^C

CHLD PROCESS 23782 closing socket desciptor 4 and exiting...
CHLD PROCESS 23752 closing socket desciptor 4 and exiting...

Main process 23737 closing socket descriptor 3 and exiting...
selma@salle231-01 $
```