Compte rendu : TP4 Socket (partie 1)

Auteur : PELLETIER Sébastien M1 Info TP1b

Instalation d'un service echo sur VM3

Apres avoir ajouter la route par default pour avoir un acces à internet avec la commande `ip route add default via 10.0.2.2` l'installation du service echo s'est faite sans souçis.
(`apt-get install inetutils-inetd; update-inetd --add "echo stream tcp nowait nobody internal"; service inetutils-inetd start; service inetutils-inetd restart`)

Essais depuis V2 :

1	0.000000000	172.16.2.131	172.16.2.163	TCP	74 50297→7 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
2	0.000231000	172.16.2.163	172.16.2.131	TCP	74 7→50297 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
3	0.000403000	172.16.2.131	172.16.2.163	TCP	66 50297→7 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=30250 TS
4	8.262364000	172.16.2.131	172.16.2.163	ECHO	81 Request
5	8.262599000	172.16.2.163	172.16.2.131	TCP	66 7→50297 [ACK] Seq=1 Ack=16 Win=29056 Len=0 TSval=229010
6	8.262625000	172.16.2.163	172.16.2.131	ECHO	81 Response
7	8.262738000	172.16.2.131	172.16.2.163	TCP	66 50297→7 [ACK] Seq=16 Ack=16 Win=29312 Len=0 TSval=32316

On remarque l'initiation de la connection (3 premieres trames) puis l'échange d'une trame allée et retour avec les ACK.

Client/serveur simples

Compilation des programmes :

- `gcc -Wall echoserveur.c -o echoserveur.exe`
- `gcc -Wall echoclient.c -o echoclient.exe`

Ctrl+c envoi le signal SIGINT au processus afin de l'interrompre.
Ctrl+d envoi EOF sur l'entrée standard.

La fermeture du client via Ctrl+d et plus propre qu'avec Ctrl+c car cela permet fermer correctement la connection TCP avec le serveur du fait que echoclient.c vérifie si stdin n'est pas fermé (EOF).

Le serveur echo (echoserveur.c) n'a pas de fonctionnalité implenté pour traiter plusieurs clients de manière « simultanée » cela a pour effet de mettre en attente le client supplémentaire jusqu'à ce que le premier client termine sa connexion.

Le serveur (echoserveur.c) écoute sur le port donné en paramètre lors de son lancement. `./echoserveur.exe 4269` mettra en écoute le serveur sur le port 4269. On peut préciser que lorsque le serveur accepte un client, il crée un nouveau socket ayant le même port destination (celui du serveur) mais un port source différent (le port d'émission du client).

En utilisant la commande `netstat -paunt` sur VM3 on obtient la liste de toutes les connexions actives en TCP ou UDP ainsi que les serveurs en écroutes. Voici la capture d'écran ou on remarque la présence d' echoserveur en écoute sur le port 1234.

```
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat        PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:35296           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:7               0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:1234            0.0.0.0:*               LISTEN      2565/echoserveur.exe
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
tcp6       0      0 :::38339                :::*                    LISTEN      -
tcp6       0      0 :::111                  :::*                    LISTEN      -
udp        0      0 0.0.0.0:46973           0.0.0.0:*                           -
udp        0      0 0.0.0.0:1003            0.0.0.0:*                           -
udp        0      0 127.0.0.1:1013          0.0.0.0:*                           -
udp        0      0 0.0.0.0:62016           0.0.0.0:*                           -
udp        0      0 0.0.0.0:68              0.0.0.0:*                           -
udp        0      0 0.0.0.0:111             0.0.0.0:*                           -
udp6       0      0 :::60817                :::*                                -
udp6       0      0 :::1003                 :::*                                -
udp6       0      0 :::36355                :::*                                -
udp6       0      0 :::111                  :::*                                -
```

Il est possible de lancer deux serveurs en même temps sur la même machine à condition de préciser deux port differents.

Autres Clients

Il n'est pas possible pour le serveur de faire la différence entre les différents clients écho. En effet le serveur ne fait que recevoir les données à partir du réseau, il ne s'occupe pas de la manière dont elles ont été envoyées (java, python ou C). Le serveur fournis un service requête → réponse, ici la requête est n'importe quelle donnée, la réponse ces même données ; ceci illustre le principe client/serveur.

API « socket » et TCP/IP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.2.131	172.16.2.163	TCP	74	46361→1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=393472 TSecr=0 WS=128
2	0.000384000	172.16.2.163	172.16.2.131	TCP	74	1234→46361 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=373566 TSecr=393472
3	0.000605000	172.16.2.131	172.16.2.163	TCP	66	46361→1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=393473 TSecr=373566
4	0.002193000	172.16.2.163	172.16.2.131	TCP	102	1234→46361 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=36 TSval=373567 TSecr=393473
5	0.002435000	172.16.2.131	172.16.2.163	TCP	66	46361→1234 [ACK] Seq=1 Ack=37 Win=29312 Len=0 TSval=393473 TSecr=373567
6	3.760033000	172.16.2.131	172.16.2.163	TCP	72	46361→1234 [PSH, ACK] Seq=1 Ack=37 Win=29312 Len=6 TSval=394413 TSecr=373567
7	3.760469000	172.16.2.163	172.16.2.131	TCP	66	1234→46361 [ACK] Seq=37 Ack=7 Win=29056 Len=0 TSval=374506 TSecr=394413
8	3.760562000	172.16.2.163	172.16.2.131	TCP	74	1234→46361 [PSH, ACK] Seq=37 Ack=7 Win=29056 Len=8 TSval=374506 TSecr=394413
9	3.761019000	172.16.2.131	172.16.2.163	TCP	66	46361→1234 [ACK] Seq=7 Ack=45 Win=29312 Len=0 TSval=394413 TSecr=374506
10	4.549526000	172.16.2.131	172.16.2.163	TCP	66	46361→1234 [FIN, ACK] Seq=7 Ack=45 Win=29312 Len=0 TSval=394610 TSecr=374506
11	4.550009000	172.16.2.163	172.16.2.131	TCP	80	1234→46361 [PSH, ACK] Seq=45 Ack=8 Win=29056 Len=14 TSval=374704 TSecr=394610
12	4.550437000	172.16.2.131	172.16.2.163	TCP	66	46361→1234 [ACK] Seq=8 Ack=59 Win=29312 Len=0 TSval=394610 TSecr=374704

Sur cet exemple, je connecte un client à un serveur echo.
On remarque donc en premier lieu, la mise en connection :

trame 1 : demande de connexion par le client (connect())
trame 2 : réponse du serveur (accept())
trame 3 : ACK du client pour confirmer la connexions

Il y a ensuite des échange de données « classiques » (send() & recev()) :

trames 4-5 : Envoi du message de bienvenu par le serveur et ACK du client
trames 6-9 : Envoi de données du client et echo de celles ci par le serveur

Enfin le client initie la fermeture de la connexion avec le serveur :

trame 10 : demande de fermeture du client (`shutdown(s, SHUT_WR)`)
trame 11-12 : ACK du serveur puis du client pour terminer la connexion