TP3 sur les sockets

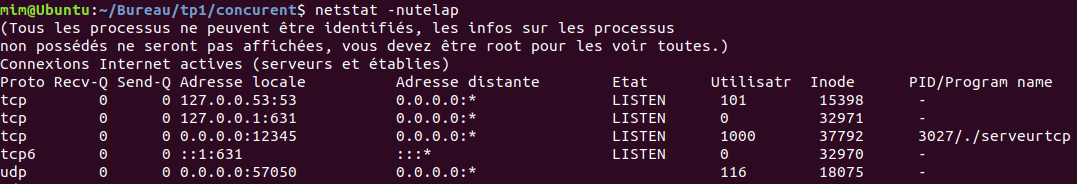
# Ecrire un serveur concurrent

## Modifier le code du serveur pour obtenir un serveur concurrent

## Testez et vérifiez avec ps –aefd que les processus fils sont bien lancés

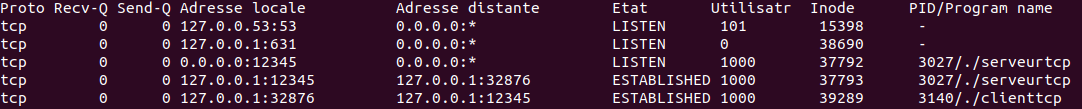
## Analysez et commentez un résultat d’execution de netstat –nutelap avant, pendant et après l’exécution de 1, 2, 3 clients

**Avant la connexion des clients :** il y a que la socket d’écoute du serveur qui est définie et en mode listen.

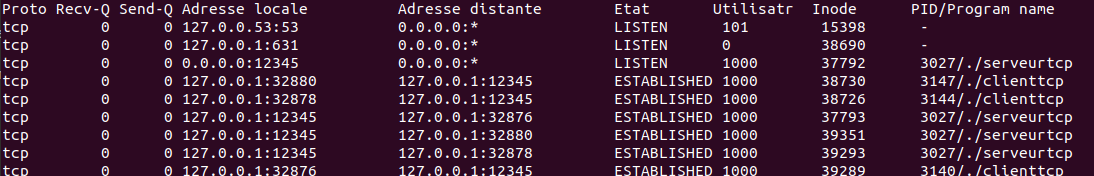


**Lorsqu’on lance les 3 clients à la fois** , la socket d’écoute du serveur reste toujours disponible, mais il y a 3 nouveau couple de sockets(Serveur=>client,client=>Serveur) qui sont défini et en mode établie chacune est liée avec un client.

Voici avec 1 seul client :

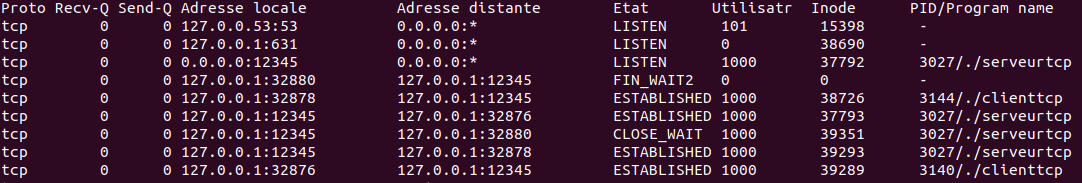


Voici quand les 3 clients sont connecté :

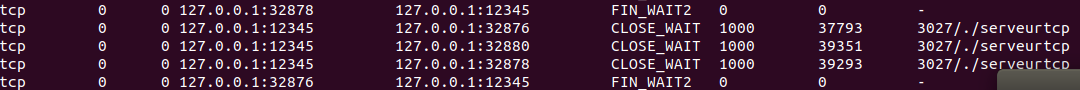


**Lors de la fin de la communication :**

ici j’ai déconnecté que un des clients , on vois que l’état de la socket (client=> serveur) est FIN , et la socket du coté serveur(serveur=> client) est en mode close.

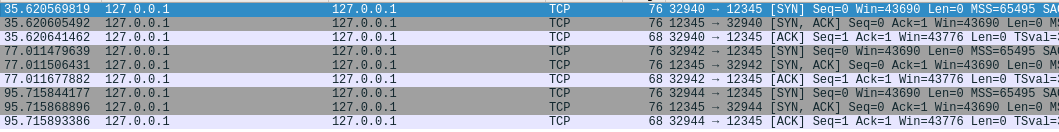


Et voila quande je déconnecte les 2 autres clients :



## Analysez et commentez une capture wireshark

**Etablissement de la connexion** : 3 blocks de 3 lignes successivement. Comme d’habitude vu que tcp est orienté connexion , alors au départ de la communication , il envoi des SYN , SYN-ACK,ACK. Pour dire que la connexion a été établie



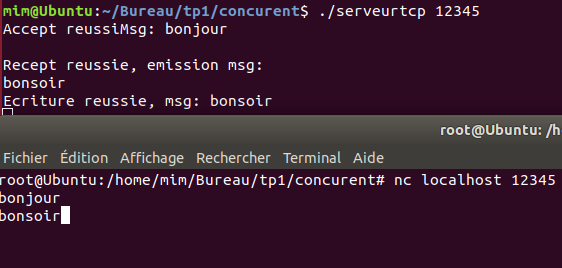
**Lors de l’échange de données** ( client envoi hello au serveur) : donc c’est un packet de type PUSH-ACK , et le serveur répond avec un ACK et dit qu’il attend le 6eme octet étant donné que hello\n fait 6octet en total, et c’est pareil pour chaque client qui envoi ou reçoit un msg au serveur



**Lors de la fermeture de la connexion :** celui qui ferme la communication envoi un FIN , l’autre répond par ACK(pour dire fin reçu) et FIN pour dire que pour lui aussi c’est bon , et puis l’autre bout répond avec un ACK pour dire j’ai reçu ta demande.



## Utilisez nc en client pour vous connecter au serveur



# Ecrire un serveur et un client udp.

## En modifiant le moins d’instructions possible écrire un serveur et un client UDP voir le code source joint

## Analysez et commentez un résultat d’execution de netstat –nutelap avant, pendant et après l’exécution de 1, 2, 3 clients

Avant la communication avec un client ,le serveur est lancé.



Lorsque le client envoi un msg au serveur, alors on voit une nouvelle ligne qui se rajoute :



## Analysez et commentez une capture wireshark

Avant que l’echange des msg, wireshark ne capture aucun paquet étant donné qu’il y a pas d’établissement ni de fermeture de connexion, vu que UDP est un protocole sans connexion.

**Lors de l’échange de paquets :**

Ici le client envoi Hello , et le serveur répond par Hi, on voit que la taille de Hello est 5octet , envoyé sur la même adresse IP du port 35308 a 12345(le serveur) ,et que la taille de HI comme réponse est 2octet envoyé sur la même adresse IP du port 12345 à 3508(le client)



Il n’y pas de paquet pour finaliser la communication.

## Lancer un serveur udp et utilisez nc pour connecter un client à ce serveur





## Analysez et commentez un résultat d’execution de netstat –nutelap avant, pendant et après l’exécution de 1, 2, 3 clients

Avant l’exécution des clients il n’y que la socket du serveur qu’est allumée :



lors de l’envoi d’un paquet du client au serveur



Lorsque le serveur répond au client : le client se déconnecte et la socket disparait de l’affichage de la commande netstat

## Analysez et commentez une capture wireshark

Avant l’exécution des clients il y aucun packet a capturé sur wireshark.

Quand un client envoi un msg au serveur :

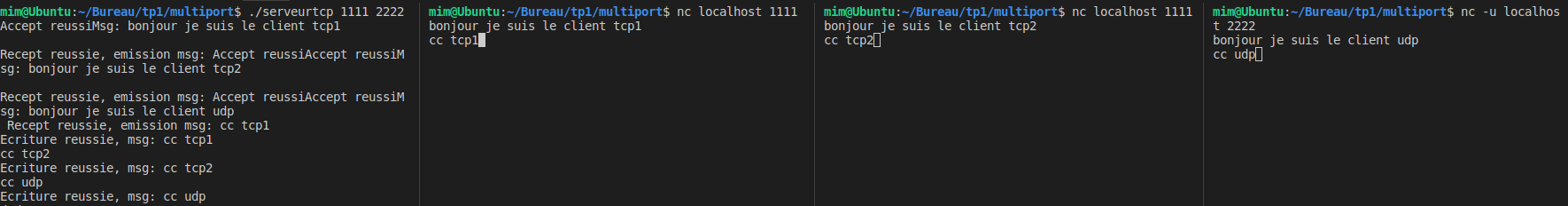


Lorsque le serveur répond au client :



# Ecrire un serveur multiport

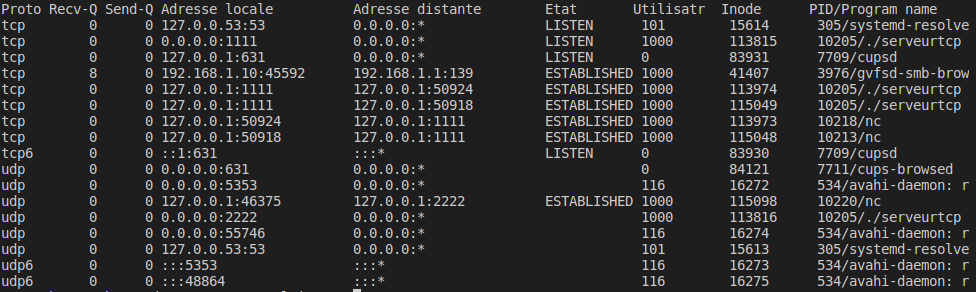
## En modifiant le moins d’instructions possible écrire un serveur capable d’attendre des messages sur 2 ports différents donnés en paramètres (il faudra utiliser la primitive select)



## Analysez et commentez un résultat d’execution de netstat –nutelap avant, pendant et après l’exécution de 1, 2, 3 clients

Lors de la connexion des 3 clients, on peut remarquer qu’il y a 4 socket établie depuis le serveur :

1. 2 établie avec les 2 clients tcp
2. 1 c’est la socket d’écoute pour tcp
3. 1 c’est la socket de UDP
4. Deux socket client ( nc ) établies
5. Une socket udp du client udp établie

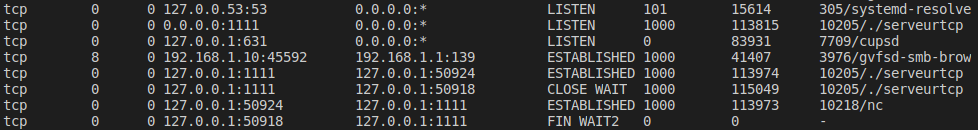


Lors de l’échange de donnée entre un client et serveur tcp il n y pas de difference qu’on voir sur la netstat parce que la connexion étant établie depuis le début.

Lors d’un échange de donnée entre un client et serveur udp : on voit que la socket du clientUdp(46375) est lié avec le serveur udp (2222)



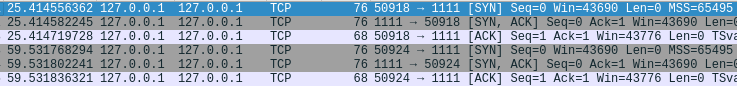
Lors de la fermeture de la connexion tcp entre tcp1 et le serveur : on peut remarqué que la socket du coté client a été fermé(FIN\_WAIT2) , et celle du serveur aussi(CLOSE WAIT)



Lors de la fermeture de la connexion udp entre le client et le serveur , il y a juste la socket du coté client qui disparait de l’affichage de netstat -nutelap

## Analysez et commentez une capture wireshark

Lors de la connexion des 3 client( 2 tcp , 1 udp ) on remarque que y que les sockets tcp qui sont établie. Udp vu que c’est un protocole sans connexion on n’a pas de trame a capturé sauf lors de l’echange de donnée.



Lors d’un envoi d’un échange de msg entre le client tcp1 et serveur : il y a comme d’habitude, des packet avec PUSH,ACK , qu’est envoyé puis la réponse en ACK pour idre que c’est reçu



Lors d’un envoi d’un échange de msg entre le client udp(46375) et le serveur (2222) :il y a auucn contrôle (seq , ack …) puisque udp ne fait pas contrôle de fiabilité.



Lors de la fermeture de la connexion tcp entre tcp1 et le serveur : ( le client lance la fermeture ), le client envoi un FIN,ACK au serveur et le serveur envoi un ACK



Lors de la fermeture de la connexion udp entre le lcient et le serveur : il n y aucune trame a capturé étant donné qu’aucune connexion n’est établie.

Voir le fichier source serveurmultiprotocole.c ci-joint avec ce document, dans le dossier multiport.