**ESIEE Paris, année 2018-2019**

**3R-IN3 Systèmes d’exploitation**

**TP n°3 Mai 2019**

**Redirections & expressions régulières**

**TRAVAIL PAR GROUPE DE DEUX ETUDIANTS**

**TRES IMPORTANT : Travail par groupe de deux étudiants.  
Le compte rendu (au format Word97 2003 ou 2007, PDF, ou texte simple) doit impérativement être envoyé par mail à cdeblang@sfr.fr avec le nom des participants du groupe dans le nom des fichiers fournis en PJ de votre mail et dans la rubrique objet du mail. Tout libellé non conforme sera pénalisé de deux points et tout envoi à une autre adresse mail ne sera pas pris en compte.**

**Le compte rendu devra être fourni en fin de séance.**

**Tout envoi après la fin de séance sera pénalisé de trois points par ¼ heure de retard.**

**Ce compte rendu comprendra le texte source de vos programmes ET la copie des écrans de l’exécution de ces programmes.**

**Nota :**

1. **Vous devez booter Linux en version 64 bits (si uname –a ne vous renvoie pas une réponse en –amd64, vous devez rebooter)**
2. **Sur Linux, l’option –E de l’utilitaire sed, qui permet de considérer les expressions régulières étendues, est rebaptisée –r. Cette remarque ne vaut que pour sed (-E conserve son sens dans grep).**

**Prérequis 1 : Rappel du fonctionnement de nc (netcat)**

**nc** est l’utilitaire netcat, qui permet d'associer une socket de communication IP (TCP ou UDP) à l'entrée ou la sortie standard du shell. Il fonctionne selon deux modes :

- un mode "client" **: nc machine port** se connecte sur la machine et le port spécifiés.

- un mode "serveur" : **nc -l -p port** écoute sur le port spécifié de la machine locale.

**Remarque** : Dans certaines distributions de Linux –l et –p sont incompatibles et le serveur est alors lancé par nc –l port. Toutes les demandes entrantes sur ce port sont alors acceptées et traitées par lui.

Dans les deux cas, le comportement de **nc** après établissement de la connexion est toujours le même :

* tout ce qui est envoyé vers son entrée standard est lu en boucle et dupliqué vers la socket de communication
* tout ce qui est envoyé vers la socket de communication est lu en boucle et dupliqué vers sa sortie standard

jusqu'à ce que l'entrée standard et la socket de communication soient fermées.

On propose de tester nc. Pour cela, ouvrez deux terminaux :

* dans le terminal "serveur", lancez

**nc -l -p 8080** **(ou nc –l 8080 cf Remarque ci-dessus)**

* puis, dans le terminal "client", lancez **nc localhost 8080**

Tapez alors du texte dans l'un ou l'autre des terminaux. Vous devriez voir la réplique de ce que vous tapez s'afficher dans l'autre terminal.

**Prérequis 2 : Utilisation de coproc**

* Copiez dans votre répertoire courant le **fichier binaire exécutable coproc** et le **script shell readwrite.sh** (qui vous ont été fournis en début de TP).
* Le script **readwrite.sh** contient le code suivant:

**#!/bin/sh**

**#**

**echo "Ici PID $$, j'écris $1" 1>&2**

**echo "$1"**

**read ligne**

**echo "Ici PID $$, j'ai lu: $ligne" 1>&2**

Lancez

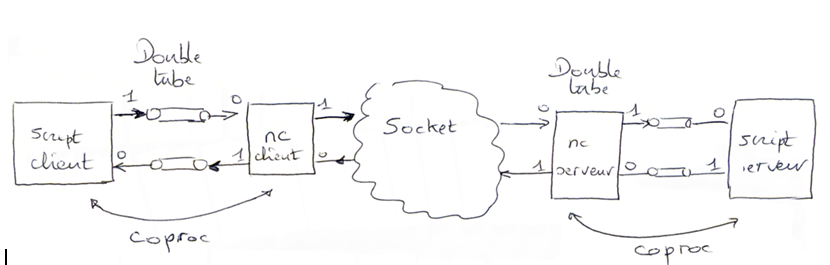
**./coproc "./readwrite.sh chaine1" "./readwrite.sh chaine2"**

en remplaçant éventuellement **chaine1 et chaine2** par d'autres valeurs (toto, titi ou autre)

**Que fait la commande coproc?   
Pourriez-vous implémenter cette commande avec un script shell ?   
Pourquoi ?**

**1) Objectif du TP**

CLIENT <-------------------------------------------------------------------------------------> SERVEUR





L’objectif du TP est d’écrire un programme fonctionnant en client/serveur, comme représenté en

figure 1. Le programme complet comporte **2 types de processus** (client et serveur).

**Le client fournit au serveur soit une adresse de IPv4, soit une adresse de réseau IPv4 de la forme aaa.bbb.cccc.ddd/p, où p ne peut valoir que 32, 24, ou 16 selon la classe d’adresse.**

**En réponse, le serveur consulte les tables ARP de la machine sur laquelle il tourne, et livre les adresses MAC et les interfaces correspondant à cette ou ces adresses, s’il en trouve.**

En résumé, le serveur offre un service de consultation à distance des tables ARP de la machine sur laquelle il tourne.

**1.1 LE SCRIPT CLIENT :**

* pose une question au serveur (sur sa sortie standard, canal 1),
* attend et lit la réponse du serveur (sur son entrée standard, canal 0),
* affiche à l’écran (sur sa sortie d’erreur standard, canal 2, car la sortie standard 1 est déjà reliée par coproc à l’entrée d’un tube) la réponse qui lui est fournie par le serveur

**La question posée par le client au serveur est une expression régulière se résolvant à au moins une adresse IPv4.** Exemples : 147.215.50.88 ou 147.215.50/88/32, 147. 215.50.0/24

**1.2 LE SCRIPT SERVEUR :**

* lit (sur l’entrée 0 du serveur) la question envoyée par le client,
* lance ARP et filtre sa sortie
* renvoie (par la sortie 1 du serveur) les lignes produites par ARP en ne conservant que l’adresse IP, l’adresse MAC, et l’interface.

**1.3 EXEMPLES**

Soit une machine accessible par l’adresse 10.0.0.1, sur laquelle /usr/sbin/arp -n donne les lignes suivantes :

Adresse TypeMap AdresseMat Indicateurs Iface

192.168.122.170 ether 52:54:00:23:45:f9 C virbr0

147.215.50.195 ether 70:5a:0f:40:1b:63 C enp7s0f0

147.215.50.223 ether 40:6c:8f:05:e6:4d C enp7s0f0

147.215.50.254 ether bc:16:65:42:2f:3f C enp7s0f0

147.215.50.11 ether 00:3e:e1:c0:21:43 C enp7s0f0

147.215.50.1 ether 00:00:0c:07:ac:05 C enp7s0f0

147.215.50.112 ether c4:34:6b:63:29:c8 C enp7s0f0

147.215.50.25 ether c4:34:6b:62:ab:7b C enp7s0f0

**Exemple 1** : le client lancé par :  
/client 10.0.0.1 147.215.50.255/24

devra afficher

147.215.50.223 40:6c:8f:05:e6:4d enp7s0f0

147.215.50.254 bc:16:65:42:2f:3f enp7s0f0

147.215.50.11 00:3e:e1:c0:21:43 enp7s0f0

147.215.50.1 00:00:0c:07:ac:05 enp7s0f0

147.215.50.112 c4:34:6b:63:29:c8 enp7s0f0

147.215.50.25 c4:34:6b:62:ab:7b enp7s0f0

**Exemple 2** : le client lancé par :  
./client 10.0.0.1 192.168.122.170

devra afficher

192.168.122.170 52:54:00:23:45:f9 virbr0

**Exemple 3** : le client lancé par :  
./client 10.0.0.1 192.168.122.1/32

ne devrait rien afficher.

NOTE : dans un premier temps, vous remplacerez 10.0.0.1 par 127.0.0.1 pour travailler sur votre propre machine, jusqu’à ce que votre client/serveur fonctionne. Mais il n’est pas interdit de vous tester entre machines et binômes en fin de séance, au contraire !

**2) Version mono-client, mono-serveur sans contrôle des paramètres du client**

On considère le cas simple où il n’y a qu’un seul client, et un seul serveur qui communiquent de manière bidirectionnelle à travers des tubes grâce à l’utilisation de **coproc et nc** (cf prérequis 1 et 2)

**AU FINAL DANS CETTE VERSION**

**Coté client, coproc** lancera conjointement

Le script client et **nc**

**Coté serveur, coproc** lancera conjointement

Le script serveur et **nc**

(nc ne servant qu’à gérer les communications bidirectionnelles entre le client et le serveur à travers la socket)

CLIENT <-----> SERVEUR

Est en fait

ScriptClient -- DoubleTubeClient -- nc **== SOCKET==** nc -- DoubleTubeServeur -- ScriptServeur

Chaque double tube est géré par un coproc

**TRAVAIL PREPARATOIRE**

**TRAVAIL 1**

1. Ecrivez un script shell **clientV1.sh** qui

* envoie sur la sortie 1 une copie de ses 2 paramètres d’appel (adresse du serveur lui-même, et adresse réseau à examiner),
* puis lit sur l’entrée 0 la réponse conformément à la description de la réponse du serveur décrite en 1.1 et 1.2)
* et affiche sur la sortie 2 une réponse conforme aux exemples fournis en 1.3

1. Ecrivez un script serveur **serveurV1.sh**  qui

* lit sur l’entrée 0 un nombre hexadécimal et une base exprimée en décimal
* puis effectue le traitement décrit ci-dessus en 1.2

1. **Testez manuellement et indépendamment** le client et le serveur directement à partir d’un terminal (l’entrée zéro est donc le clavier et la sortie 1 est l’écran)

**Fournir :   
- le code de vos deux scripts, clientV1.sh et serveurV1.sh,**

**- les lignes de commande coté client et coté serveur,  
- une copie des écrans d’exécution du client et du serveur.**

**TRAVAIL 2**

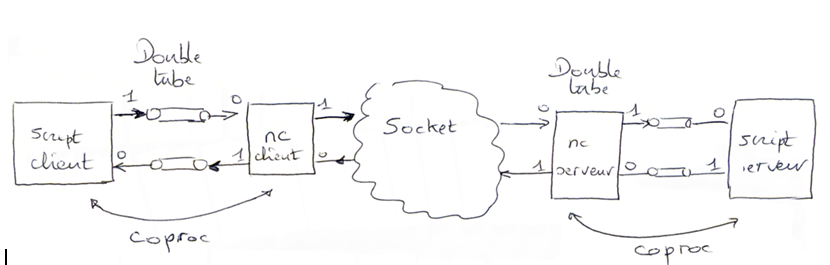
Lancez le client et le serveur en les reliant par un double tube à l’aide de coproc sans utiliser de communication réseau

ScriptClient -- DoubleTube -- ScriptServeur

**Fournir :   
- la ligne de commande de mise en œuvre,   
- une copie des écrans d’exécution.**

**TRAVAIL 3**

Donnez finalement la ligne de commande coté client et la ligne de commande coté serveur qui permettent de faire communiquer le serveur et le client en mettant en oeuvre le schéma final avec une seule question et une seule réponse.



**Fournir :   
- les lignes de commande coté client et coté serveur,  
- une copie des écrans d’exécution du client et du serveur.**

**3) Version mono-client, mono-serveur avec contrôle des paramètres par le client**

Le fonctionnement du serveur et la mise en œuvre globale client/serveur demandée dans le travail 3 précédent sont inchangés.

**Mais dans cette version le client est plus compliqué que dans 2).**

Le client :

* Vérifie à l’aide d’une expression régulière que les deux paramètres sont bien conformes au format décrit en 1.1
  + Si les paramètres ne sont pas conformes, **aucune donnée n’est pas transmise au serveur**, et un message d’erreur est affiché à l’écran par la sortie 2
  + Si les paramètres sont conformes, **ils sont transmis au serveur** et le client affiche à l’écran par la sortie 2 la réponse qui lui est fournie par le serveur conformément aux exemples du 1.3.
* Programmez **clientV3.sh correspondant (le serveur étant inchangé par rapport à 2)**

**Fournir :   
- le code du script clientV3.sh,**   **- les lignes de commande coté client et coté serveur,  
- une copie des écrans d’exécution du client et du serveur.**

**4) Travail facultatif : Version multi-clients, mono-serveur avec contrôle de la saisie par le client**

1. Expliquez les modifications que vous devez apporter au 3) pour obtenir non pas un, mais N clients distincts sur un seul serveur..
2. Programmez **clientV4.sh et serveurV4.sh** correspondant au a)
3. Quel problème peut se poser si les clients s’exécutent en parallèle ?

**Fournir :   
- le code de vos deux scripts, clientV4.sh et serveurV4.sh,**

**- les lignes de commande coté client et coté serveur,  
- une copie des écrans d’exécution du client et du serveur.**