


Introduction aux systèmes d'exploitation (IS1)

TP n° 10 : enchaînement de commandes

Lors du TP n° 8, nous avons vu comment lancer plusieurs commandes *en parallèle*, en connectant leurs sortie et entrée standard à l'aide de tubes. Il est également possible de lancer plusieurs commandes en parallèle sans redirection de leurs flots standard, simplement en les lançant en arrière-plan (sauf éventuellement la dernière) à l'aide du caractère « & ». Au cours de ce TP, nous allons voir plusieurs manières de lancer des commandes *en série*, c'est-à-dire l'une après l'autre, sans attendre l'invite du shell entre les commandes.

Modalités de rendu Comme d'habitude, il vous est demandé de déposer sur Moodle un fichier appelé `reponses_TP10.txt` contenant les commandes utilisées pour répondre aux questions marquées par le symbole . N'oubliez pas d'y insérer vos nom(s) et prénom(s), surtout si vous travaillez en binôme.

L'enchaînement simple La première méthode consiste à lister les différentes commandes à exécuter en utilisant le connecteur « ; » pour les séparer :

commande₁ ; commande₂ ; ... ; commande_n





Le shell exécute alors *commande₁*, puis, une fois celle-ci terminée, *commande₂*, et ainsi de suite jusqu'à la dernière commande.

Exercice 1 – enchaînement simple vs exécution en parallèle

1. Comparer l'exécution des deux lignes de commande suivantes :

`sleep 5 ; echo "bouh!"` et `sleep 5 & echo "bouh!"`

Pour comprendre le déroulement de ces instructions, ouvrez un deuxième terminal depuis lequel vous surveillerez la naissance et la terminaison des processus créés. Pour avoir le temps de les observer, nous allons augmenter la durée des deux commandes.

2. Exécuter « `sleep 500 ; xclock` ».  Combien de processus ont été créés ? Tuer le processus qui exécute « `sleep 500` ». Que se passe-t-il alors ? Terminer maintenant le nouveau processus.
3.  Mêmes questions en recommençant avec « `sleep 500 & xclock` ».
4. Exécuter « `(sleep 500 ; xclock) &` ».  Déterminer la généalogie des processus créés. Tuer le processus qui exécute « `sleep 500` ». Que se passe-t-il ? Tuer le processus « `xclock` ». Que reste-t-il ?
5. Exécuter à nouveau l'instruction précédente, puis tuer le père du processus qui exécute « `sleep 500` ».  Que se passe-t-il cette fois ?

Valeur de retour d'un processus Tout processus UNIX renvoie à son processus père une *valeur de retour* indiquant les conditions de son arrêt. Cette valeur est un entier positif, par convention égal à 0 **si et seulement si** l'exécution et la terminaison se sont déroulées correctement. Les autres valeurs de retour possibles d'une commande sont documentées dans son manuel.

Exercice 2 – « *echo \$?* »

La variable d'environnement « ? » contient la valeur de retour de la précédente commande exécutée par le shell.

1. Exécuter « *ls* ». ➤ Quelle est la valeur de retour du processus ? Recommencer avec un nom de fichier qui n'existe pas, par exemple « *ls grosminet* ».
2. Exécuter « *sleep 2* », en laissant l'exécution aller à son terme. Quelle est la valeur de retour du processus ?
3. Exécuter « *sleep 100* », et interrompre le processus à l'aide de *ctrl-C*. ➤ Quelle est sa valeur de retour ? Recommencer en interrompant le processus à l'aide de différents signaux.

Les enchaînements conditionnés Deux autres connecteurs permettent d'exécuter des commandes les unes après les autres :

commande₁ && commande₂ && ... && commande_n

Le shell exécute alors *commande₁*, puis, une fois celle-ci terminée, si sa valeur de retour est nulle, il exécute *commande₂*, et ainsi de suite jusqu' à la dernière commande de la liste. Cette méthode permet d'enchaîner les commandes ***tant que*** tout se déroule correctement.

commande₁ || commande₂ || ... || commande_n

Le shell exécute *commande₁*, puis, une fois celle-ci terminée, si sa valeur de retour est non nulle, il exécute *commande₂*, et ainsi de suite jusqu' à la dernière commande de la liste. Cette méthode permet d'enchaîner les commandes ***jusqu'à ce qu'***une d'entre elles s'exécute sans erreur.

Exercice 3 – enchaînements conditionnés

1. Exécuter « *sleep 5 || xclock* », puis « *sleep 5 && xclock* », sans interrompre la commande « *sleep* ». ➤ Comparer, puis tuer le(s) processus qui reste(nt).
2. Exécuter ensuite « *sleep 500 || xclock* », puis « *sleep 500 && xclock* », en interrompant la commande « *sleep* » par l'envoi de divers signaux. ➤ Comparer les différentes réactions, puis tuer le(s) processus qui reste(nt).

Exercice 4 – comparer deux fichiers

Dans cet exercice, nous allons utiliser la commande « cmp », qui compare deux fichiers octet par octet.

1. Créer deux fichiers `pareil` et `memme` au contenu identique, et un fichier `different` au contenu différent. ➤ Comparer le comportement et la valeur de retour de la commande « cmp » selon qu'elle a été appelée avec deux fichiers identiques ou avec deux fichiers différents.
2. ➤ Le message affiché par la commande « cmp » lorsque les deux fichiers sont différents correspond-il à la sortie standard ou à la sortie erreur standard ?
3. Écrire une séquence d'instructions qui compare deux fichiers et affiche
 Les deux fichiers sont identiques.
(uniquement) quand c'est le cas.
4. ➤ Même question en faisant en sorte que seule la phrase demandée s'affiche, et pas le message associé à la commande « cmp ».
5. ➤ Écrire une séquence d'instructions qui affiche le message
 Les deux fichiers sont différents.
(uniquement) quand c'est le cas.
6. ➤ Combiner ces deux séquences pour afficher la phrase correcte en fonction du résultat de la commande, sans afficher le message associé à « cmp ». Indice : vous pouvez délimiter une séquence d'instructions à l'aide de parenthèses.