TD/TP N°3 Linux

Module : Administration Système Linux
Filière : Cycle d'ingénieur Génie Informatique
Niveau : GI2
Les processus

On appelle *processus* un objet dynamique correspondant à l'exécution d'un programme ou d'une commande Unix. Cet objet regroupe plusieurs informations, en particulier l'état d'avancement de chaque programme, l'ensemble des données qui lui sont propres, ainsi que d'autres informations sur son contexte d'exécution.

Exercice 1. *Lister les processus :* ps.

1. Lancez quelques programmes depuis l'interface graphique KDE, puis lancez les utilitaires xclock et xcalc depuis votre terminal (n'oubliez pas de faire suivre leurs noms par le caractère & pour ne pas perdre la main dans le terminal).
2. Sans fermer les programmes que vous venez de lancer, testez la commande ps depuis le terminal. Lancez un second terminal et recommencez. Décrivez les changements. Puis fermez le second terminal.
3. L'option -l de ps permet d'afficher plus de détails sur les processus. Testez cette option et repérez les caractéristiques précédemment évoquées. Détaillez, en vous aidant, au besoin du manuel, la signification des colonnes UID, PRI, TT, VSZ, TIME.
4. Comment peut-on afficher la liste de tous les processus du système (notamment ceux dont vous n'êtes pas le propriétaire, et ceux qui ne dépendent pas d'un terminal) ?
5. Comment peut-on obtenir plus d'informations sur les processus listés par ps ? (Il y a deux réponses possibles selon les informations supplémentaires que l'on veut afficher.)
6. Comment peut-on obtenir la liste des processus dont le propriétaire est le *super utilisateur* ?

Contrôler l'exécution d'un processus

Les programmes ne fonctionnent malheureusement pas toujours comme prévu. Un garant important de la stabilité du système est donc de pouvoir mettre fin à l'exécution de processus devenus instables ou ne répondant plus. Il existe plusieurs méthodes pour mettre fin à des processus récalcitrants.

Exercice 2. *Signaux.*

La commande kill permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas *seulement* à "tuer" un processus. Il existe de nombreux signaux différents dont vous trouverez la signification dans le man. Les signaux les plus courants sont SIGTERM et SIGKILL, qui servent à terminer un processus.

1. Comment peut-on obtenir la liste de tous les signaux que l'on peut envoyer aux processus ? (Indication : la réponse se trouve dans la page de manuel de la commande kill). Quels sont les numéros correspondant aux signaux SIGTERM, SIGKILL, SIGSTOP et SIGCONT ?
   La syntaxe d'envoi d'un signal est : kill *-signal pid* où *signal* est un numéro ou un nom de signal (le signal par défaut est SIGTERM).
2. Testez les signaux mentionnés ci-dessus sur un processus xclock préalablement lancé, et décrivez le résultat.
3. Lancez un second terminal, puis testez les signaux SIGTERM et SIGKILL sur le processus correspondant à ce nouveau terminal. Que constatez-vous ? Qu'en déduisez-vous sur le fonctionnement des signaux SIGTERM et SIGKILL ?
4. Repérez parmi les processus actifs un processus dont vous n'êtes pas propriétaire, et tentez de le stopper à l'aide du signal SIGSTOP. Que se passe-t-il et qu'en déduisez-vous ?

Gérer les tâches dans le shell : *jobs*

Lorsque des processus sont lancés depuis un terminal, certains shells modernes et en particulier celui que vous utilisez (bash), fournissent un ensemble de mécanismes pour gérer leur exécution. Dans ce contexte, on parle de tâches (*jobs*).

Exercice 3. *Avant et arrière-plan.*

1. Depuis un terminal, lancez un processus xclock et un processus konqueror **avec un** & **final**. Les fenêtres correspondantes s'affichent, et vous ne perdez pas la main sur le terminal (l'invite réapparaît immédiatement).

On dit que le processus que vous venez de lancer est à l'*arrière-plan* du shell (*background job*). Le shell vous indique alors le PID du processus qui vient d'être déclenché en affichant par exemple : [3] 31926 où 31926 est le PID.

2. Depuis un terminal, lancez un processus xclock **sans le** & **final**. On dit que le processus que vous venez de lancer est à l'*avant-plan* du shell (*foreground job*).
3. Depuis le terminal, pressez la combinaison de touche Ctrl-Z, aussi appelée commande de suspension (*suspend*). Quel est le résultat ? Pouvez-vous encore utiliser xclock? On dit que le processus xclock est *suspendu*.
4. La commande Ctrl-Z correspond à l'envoi d'un signal via la commande kill. Quel est ce signal ? Vérifiez vos dires par l'expérience. Pour simplifier la manipulation de plusieurs processus dépendant d'un même shell, il leur est attribué un numéro de tâche (*job number*) propre au shell qui les contrôle (différent en particulier du PID). La commande jobs permet d'afficher une liste de ces processus triée par numéro de job, ainsi que leur état actuel (suspendu ou en cours). Un signe "+" marque le job courant, un "-" le job précédent.
5. Testez cette commande, et comparez sa sortie à celle de ps sans argument.
6. Exécutez xclock & puis xemacs & puis jobs. Que constatez-vous sur les numéros de job associés aux programmes ? Nous sommes maintenant capables de lancer des processus à l'avant ou à l'arrière-plan et de suspendre des processus. Deux commandes supplémentaires permettent de ramener un processus à l'avant-plan (fg) ou de faire reprendre son exécution à l'arrière-plan à un processus interrompu (bg). Sans argument, ces commandes s'appliquent au processus courant (cf. question précédente), elles peuvent aussi être suivies d'un argument de la forme %n, où n est un numéro de job.
7. Testez les commandes fg et bg pour faire successivement passer à l'avant-plan et à l'arrière-plan les jobs que vous avez lancés. Contrôlez l'état de vos processus avec la commande jobs.
8. A quoi est équivalente la séquence : xclock – Ctrl-Z – bg ?
9. Terminez chacun de vos jobs en les ramenant à l'avant-plan et en pressant la combinaison de touches Ctrl-C. A quel signal correspond ce raccourci ?
10. Trouvez comment relancer un processus interrompu avec la commande kill. **Valeurs de retour** Dans l'architecture Unix, un processus qui se termine communique en principe à son environnement, et en particulier à son processus père, une information sur les conditions de son arrêt, que l'on appelle *valeur de retour*. Par convention, une valeur de retour nulle (égale à 0) signifie que l'exécution et la terminaison du processus se sont déroulées normalement. Une valeur strictement positive correspond à un cas particulier ou à une erreur. Les valeurs de retour possibles pour les différentes commandes sont en général documentées dans la page de manuel correspondante.

Exercice 4. Afficher la valeur de retour.

Dans un terminal, la commande echo $? permet d'afficher le code de retour de la commande précédente1.

1. Lisez le manuel de la commande cmp. A quoi sert cette commande ? Quels sont ses valeurs de retour possibles (il y en a au moins 3) ?
   Cette notation sera expliquée en détail dans le TP sur les variables du shell.
2. Donnez un cas d'utilisation de cmp pour chaque type de valeur de retour qui figure dans le manuel (et vérifiez, à chaque fois, la valeur de retour).
3. Une fois la question précédente terminée, tapez à nouveau echo $?. Le résultat vous paraît-il normal ? Pourquoi ? Comment expliqueriez-vous ce résultat ?

Enchaîner plusieurs commandes

Le shell bash est un véritable langage de programmation (ou plus précisément un *langage de scripts*, possédant ses variables, structures de contrôle (conditionnelles, boucles, cas), fonctions, etc. Au fur et à mesure des séances, nous découvrirons certains de ces aspects. La construction la plus simple dans tout langage de programmation est l'enchaînement séquentiel des commandes. En bash, il existe plusieurs possibilités pour lancer plusieurs commandes l'une après l'autre :

– *commande*1 ; *commande*2 . . . ; *commande*n

Une suite de commandes séparées par un point virgule est exécutée par le shell de la manière suivante :* Le shell exécute la commande 1.

* Une fois que celle ci est terminée, et quelle que soit sa valeur de retour, il exécute la commande 2 .

* Et ainsi de suite... jusqu' à la dernière commande de la liste.

– *commande*1 && *commande*2 . . .&& *commande*n

Une suite de commandes séparées par && ("et") est exécutée par le shell de la manière suivante :

* Le shell execute la commande 1.

* Une fois que celle ci est terminée, et si la valeur de retour est nulle, il exécute la commande 2 .

* Et ainsi de suite... jusqu' à la dernière commande de la liste.

Cette méthode permet d'enchainer les commandes seulement si tout se déroule correctement.

– *commande*1 || *commande*2 . . .|| *commande*n

Une suite de commandes séparées par || ("ou") est exécutée par le shell de la manière suivante :

* Le shell execute la commande 1.

* Une fois que celle ci est terminée, et si la valeur de retour est non nulle, il exécute la commande 2
* Et ainsi de suite... jusqu' à la dernière commande de la liste.

Cette méthode permet d'enchaîner les commande jusqu'à ce qu'une d'entre elles s'exécute sans erreur.

Exercice 5. *Enchaînements imposés.*

1. Ecrivez une séquence d'instructions qui compare deux fichiers de votre répertoire personnel et affiche "les deux fichiers sont identiques" le cas échéant. (Indication : utilisez la commande cmp)
2. Réciproquement, écrivez une commande qui affiche "les deux fichiers sont différents" quand c'est le cas.
3. Combinez ces deux séquences pour afficher la phrase correcte en fonction du résultat de la commande (indice : vous pouvez délimiter une séquence d'instructions à l'aide de parenthèses). Expliquez pourquoi votre solution fonctionne.