Introduction aux systèmes d'exploitation (IS1)

TP 3 – Manipulation de processus

On appelle *processus* un objet dynamique correspondant à l'exécution d'un programme ou d'une commande Unix. Cet objet recense 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.

Connaître les processus en cours d'exécution

Parmi les caractéristiques que possède tout processus, on trouve en particulier :
- son numéro d'identification (PID ou *process identifier*), qui est un nombre entier positif ;
- l'identifiant du processus qui lui a donné naissance, ou processus parent, appelé PPID (*parent PID*) ;
- son propriétaire, en général (mais pas toujours) l'utilisateur qui l'a lancé ;
- éventuellement le terminal dont il dépend (s'il existe) ;
- son répertoire courant, sa priorité de travail, son temps d'exécution, etc.

Plusieurs outils permettent d'afficher la liste des processus et un certain nombre de leurs caractéristiques. La commande `pstree` sans argument affiche l'ensemble des processus en cours, en les reliant selon leur parenté.

Exercice 1 – *Arborescence des processus :* `pstree`.

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 leur nom par le caractère `&` pour ne pas perdre la main dans le terminal).
2. Testez la commande `pstree`. Repérez dans l'arbre les processus correspondant aux programmes que vous venez de lancer.
3. Quelles options de `pstree` permettent d'afficher le propriétaire de chaque branche de processus ? Le PID de chaque processus ?

Des informations plus précises sur les processus peuvent être obtenues à l'aide de la commande `ps`.

Exercice 2 – *Détails des processus :* `ps`.

1. Sans fermer les programmes que vous avez lancés à l'exercice précédent, lancez `ps` sans argument depuis le même terminal. Lancez un second terminal et recommencez. Que constatez-vous ? Fermez le second terminal.

2. L'option `-l` de `ps` permet d'afficher plus de détails sur les processus. Testez cette option et essayez de repérer les caractéristiques précédemment évoquées. Consultez au besoin le manuel de la commande pour connaître la signification de chaque colonne (certaines, comme WCHAN ou F, peuvent être ignorées).
3. Comme pour la plupart des commandes Unix, il est possible d'afficher une aide concise sur `ps` grâce à l'option `--help`. Utilisez-la pour trouver comment afficher la liste de tous les processus du système. Profitez-en pour tester d'autres options.

Pour un affichage dynamique et interactif de l'ensemble des processus, un troisième outil intéressant est la commande `top`. C'est en quelque sorte l'équivalent Unix du « gestionnaire de tâches » de Microsoft® Windows.

Exercice 3 – *Liste actualisée des processus.*

1. Lancez la commande `top`. Dans quel ordre les processus affichés sont-ils classés ?
2. Quand `top` est actif, un appui sur la touche `u` permet de sélectionner uniquement les processus appartenant à un utilisateur donné. Affichez uniquement vos processus.
3. Cet utilitaire offre un grand nombre de fonctionnalités, comme celle de modifier la colonne de tri, sélectionner les colonnes à afficher, etc. Explorez la page de manuel et testez certaines de ces options.

Contrôler l'exécution d'un processus

Les programmes ne fonctionnent 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 4 – *Signaux.*

La commande `kill` permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, et même si c'est son usage principal, elle ne sert pas seulement à « tuer » un processus. Les signaux les plus courants sont SIGTERM et SIGKILL, qui servent à terminer un processus. D'autres signaux fréquents sont SIGSTOP et SIGCONT.

1. La liste des signaux que l'on peut envoyer aux processus s'obtient grâce à l'option `-l` de `kill`. Repérez les numéros des signaux mentionnés ci-dessus.
2. 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).

   Testez les signaux mentionnés ci-dessus sur un processus xclock préalablement lancé (utilisez l'une des commandes précédentes pour accéder à son PID).
3. Lancez un second terminal. Repérez son identifiant de processus, puis testez les signaux SIGTERM et SIGKILL sur ce processus. Que constatez-vous ? Ce comportement illustre le fait que dans certains cas, être « poli » ne suffit pas : SIGTERM demande au processus de s'arrêter (ce qu'il peut refuser), tandis que SIGKILL demande au système de l'achever.

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. Qu'en déduisez-vous ?

Exercice 5 – *Tuer un processus avec style.*

Il existe au moins deux autres techniques pour terminer des processus (sans compter le fait d'appuyer sur le bouton « fermer » dans le cas d'un processus fenêtré).

1. Lancez deux ou trois processus quelconques depuis le terminal ou l'interface graphique. Dans un terminal, lancez la commande `top`.

   La touche `k` vous permet d'indiquer que vous souhaitez terminer un processus. Il vous est ensuite demandé de désigner un processus par son identificateur, puis de spécifier un signal à lui envoyer. C'est en quelque sorte un `kill` interactif.

   Terminez ainsi les programmes que vous venez de lancer.
2. Dans certains cas, il est plus aisé de désigner un processus non pas par son identifiant mais en le pointant avec la souris. Pour ce faire, il existe la commande `xkill`. Son utilisation est assez intuitive... Testez cette commande.

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 6 – *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 `kwrite` **sans le** `&` **final**. Kwrite fonctionne mais vous perdez la main sur le terminal (l'invite ne réapparaît pas).

   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 Kwrite ?

   On dit que le processus `kwrite` est *suspendu*.

4. La commande `Ctrl-Z` correspond à l'envoi d'un signal SIGTSTP aux éventuels processus d'avant-plan. Il est donc aussi possible d'envoyer explicitement ce signal grâce à la commande `kill`. Suspendez le processus de konqueror de cette façon, et constatez l'effet sur le programme.

5. 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.

   Testez cette commande, et comparer sa sortie à celle de `ps` sans argument.

   Exécutez `xclock &` puis `jobs`. Que constatez-vous sur le numéro du job associé au programme ?

6. 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.

   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`.

7. A quoi est équivalente la séquence : `xclock` – `Ctrl-Z` – `bg` ?

8. Terminez chacun de vos jobs en les ramenant à l'avant-plan et en pressant la combinaison de touches `Ctrl-C` (correspondant à l'envoi du signal SIGINT).

9. Trouvez comment relancer un processus interrompu avec la commande `kill`.

Exercice 7 – *Dégâts collatéraux.*

La commande `sleep` se contente d'attendre le temps qu'on lui indique. Par exemple `sleep 1m` attend une minute.

1. Lancez un terminal et exécutez-y la commande `sleep 10m &`, relevez son identifiant puis fermez le terminal en cliquant sur l'icône.

2. Avant la fin des 10 minutes, affichez dans un autre terminal les informations sur le programme `sleep`. Que constatez-vous ?

3. Lorsque le terminal auquel un processus est attaché est clos ainsi, les processus reçoivent le signal SIGHUP. Afin de protéger un programme contre ce signal, il est alors possible d'utiliser la commande `nohup`. À l'aide de sa page de manuel, utilisez cette commande pour laisser `sleep` survivre après la fermeture du terminal.

4. En affichant les PID avec la commande `pstree`, repérez où est placé `sleep` avant et après la fermeture du terminal. Qu'est-ce qui change ?

   On dit alors que le processus `sleep` est orphelin.

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 8 – *Afficher la valeur de retour.*

Dans un terminal, la commande `echo $?` permet d'afficher le code de retour de la commande précédente[1].

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) ?
2. Lancez `cmp` de plusieurs façons possibles afin de couvrir tous les types de valeurs de retour qui figurent dans le manuel (utilisez `echo` comme indiqué précédemment pour afficher la valeur de retour).
3. Une fois la question précédente terminée, tapez à nouveau `echo $?`. Le résultat vous paraît-il normal ?

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 un par un chacun 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 :

– `;` – une liste de commandes de la forme

$$commande_1 \; ; \; commande_2 \ldots ; \; commande_n$$

est exécutée par le shell en lançant d'abord $commande_1$, puis chaque commande successive est lancée une fois que la précédente s'est terminée, quelle que soit la valeur de retour de cette dernière ;

– `&&` (« et ») – une liste de commandes séparées par `&&` fonctionne de façon similaire, à la différence qu'une commande n'est exécutée que si toutes les commandes la précédant ont fourni une valeur de retour nulle ;

– `||` (« ou ») – dans une liste de commandes séparées par `||`, une commande n'est exécutée que si toutes les commandes la précédant ont fourni une valeur de retour **non nulle**.

[1] Cette notation sera expliquée en détail dans le TP sur les variables du shell.

Exercice 9 – *Enchaînements imposés.*

1. Écrire une séquence d'instructions qui compare deux fichiers de votre répertoire personnel et affiche « les deux fichiers sont identiques » le cas échéant.
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).

   Pouvez-vous expliquer pourquoi votre solution fonctionne ?