L1 TDo4 *Introduction aux systèmes d'exploitation*

Amine Najahi [amine.najahi@univ-perp.fr]

Exercice 1 Processus : création, observation et terminaison

1. Trouvez l'utilité de la commande **[ps]**. La commande **[ps -u moi]** affiche les processus dont l'`UID` est moi. Affichez les processus dont vous êtes l'`UID`.
2. L'option **[-l]** de **[ps]** permet d'afficher plus d'informations sur chaque processus. Affichez avec plus de détails les processus dont vous êtes l'`UID`.
3. Que fait l'option **[-e]** de **[ps]** ?
4. A partir des colonnes `PID` et `PPID` (colonnes 4 et 5) de l'affichage de **[ps -e -l]**, reconstituez la généalogie des processus lancés depuis la fenêtre terminal (le `PID` de cette dernière est identifiable à partir de la colonne `TTY`).
5. Ouvrez un nouveau terminal. Quel nouveau processus est créé ? De qui est-il le fils ? Lancez un processus comme **[dc]** dans cette nouvelle fenêtre. Quels sont les processus lancés depuis la fenêtre ? Fermez le processus **[dc]** avec Ctrl + D. Qu'est devenu le processus **[dc]** ?

*La commande **[kill]** permet d'envoyer des signaux aux processus. Contrairement à ce que peut laisser penser le nom de la commande, tous ces signaux ne tuent pas les processus cibles (certains sont même ignorés par les processus). Un exemple d'utilisation de **[kill]** est donné par **[kill -9 1234]** qui envoie le signal 9 (appelé **SIGKILL**) au processus 1234. L'action de **SIGKILL** est de tuer le processus immédiatement. Notez que le raccourci clavier Ctrl + c dans un terminal permet d'envoyer un signal **SIGINT** au processus en cours d'exécution. L'effet par défaut d'un **SIGINT** est de tuer le processus cible. Mais contrairement à **SIGKILL**, ce signal peut être masqué par les processus auquel cas, il ne cause pas forcément la terminaison de sa cible.*

6. Dans la fenêtre, saisissez le programme (infini) shell suivant: **[while [1]; do echo bonjour; done]**. Dans une autre fenêtre, observez les processus. Y a-t-il un processus associé au programme shell précédent ? Comment peut-on stopper l'affichage infini ?
   1. Depuis la fenêtre où il a été lancé.
   2. Depuis une autre fenêtre.

   Quelle conséquence pour la fenêtre dans les deux cas ?
7. Lancez **[echo "while [1]; do sleep 1; done" > dormir]**. Puis **[chmod u+x dormir]**. Que font ces commandes ?
8. Exécutez **[./dormir &]**. Que se passe-t-il ? Observez le processus associé à **[dormir]** avec **[ps -l]**. Attention, ce dernier ne s'appelle pas `dormir`.
9. A quoi correspondent les deux processus `bash` ? Essayez de tuer le processus **[sleep]**. Pourquoi le système répond-il Aucun processus de ce type ? Fermez la fenêtre. Depuis une autre fenêtre, observez si le processus lancé par **[dormir]** a disparu.

10. Dans une nouvelle fenêtre, lancer **[**`nohup ./dormir &`**]**. Fermez la fenêtre. Le processus ayant lancé `dormir` a-t-il disparu ? Qui est son père désormais ? À quel terminal est-il lié (colonne `TTY`) ?
11. En déduire comment on lance un travail qui se poursuit après déconnexion.

Exercice 2 Redirections avec les symboles <, >, 2> et ≫

Par défaut, chaque processus possède 3 flux standards:

1. *Le flux d'entrée standard que l'on peut rediriger avec le symbole* `<`.
2. *Le flux de sortie standard que l'on peut rediriger avec le symbole* `>` *ou* `1>`.
3. *Le flux d'erreur standard que l'on peut rediriger avec le symbole* `2>`.

1. Créez un fichier `fic1` avec **[**`echo "une ligne" > fic1`**]**. Affichez le avec **[**`cat`**]**.
2. Copiez le fichier `fic1` dans `fic2` en utilisant la commande **[**`cat`**]** et le symbole `>`.
3. Lancez la commande **[**`cat < fic1 ≫ fic2`**]**. Vérifiez de nouveau contenu de `fic2`. Expliquez ce que fait la commande.
4. Créez un fichier de plusieurs lignes avec **[**`cat`**]**.
5. Récupérez la sortie de **[**`ps -e`**]** dans un fichier `fic3`.
6. Lancez la commande **[**`ls -l fichier_inexistant`**]**. Que se passe-t-il ?
7. Exécutez la commande **[**`ls -l fichier_inexistant > sortie`**]**. Que contient le fichier `sortie` ?
8. Que faut-il changer à la commande précédente pour que `sortie` contienne effectivement le résultat de la commande ?

Exercice 3 Tubes: enchaînement de commandes

1. Exécutez la commande **[**`ls -l /usr/bin > liste`**]**. Quel effet a cette commande ?
2. Trouvez à quoi sert la commande **[**`wc`**]** ? Quelle est l'effet de l'option **[**`-l`**]** de **[**`wc`**]** ?
3. Utilisez **[**`wc`**]** avec la l'option appropriée et le fichier `liste` en argument pour trouver le nombre d'entrée dans `/usr/bin`.

Le caractère `|` *(appelé tube en français et pipe en anglais) est interprété de manière spéciale par le shell. Il permet de rediriger la sortie d'une commande vers l'entrée de la commande suivante. Pour se faire, il doit être utilisé de la façon suivante* **[**`commande1 | commande2`**]**.

4. Effacez le fichier liste. Ensuite, utilisez de nouveau **[**`ls -l`**]** et **[**`wc`**]** pour trouver le nombre d'entrées dans `/usr/bin` sans passer par un fichier intermédiaire.
5. Comptez le nombre de processus existants sans passer par un fichier intermédiaire.
6. Que fait la commande **[**`find . -printf '%s %p\n'`**]** ?
7. Utilisez **[**`man sort`**]** pour trouver l'utilité de **[**`sort`**]** ainsi que son option **[**`-gr`**]**. Faites de même pour la commande **[**`head`**]**.
8. En déduire des deux questions précédentes une ligne de commande qui liste les 10 plus gros fichiers d'une arborescence.