Système d'exploitation

Contrôle

Novembre 2024

Exercise 1 *Système de fichier*

Une variation de `ls -l` (à savoir `ls -og --time-style=+''`, qui permet de ne garder que ce qui nous intéresse ici) est exécutée par quelqu'un dans un répertoire. Cela donne :

```
total 8
-rwxr-x--x 2    0  a
-rw-rw-r-- 1    0  b
lrwxrwxrwx 1    1  c -> b
drwxr-xr-x 2 4096  d
-rwxr-x--x 2    0  e
prw-r--r-- 1    0  f
lrwxrwxrwx 1    1  g -> b
drwxr-xrwx 5 4096  i
lrwxrwxrwx 1    1  j -> h
```

Analysez la sortie de la commande, puis répondez aux questions.

1. Donnez pour chacun des fichiers listés son type. [1]
2. À quelle(s) condition(s) un utilisateur peut-il : [2]
   a) lire le fichier `./e` ?
   b) écrire dans le fichier `./f` ?
   c) exécuter le fichier `./b` ?
   d) créer un fichier dans `./i/` ?
3. Dans la sortie affichée plus haut, à quoi correspond la 3e colonne (celle contenant les nombres 1, 0 et 4096) ?
4. À quoi correspond la 2e colonne (celle contenant les nombres 1, 2 et 5) ?
5. Le chemin `./d/subd/bla` a-t-il une chance d'être valide ?
6. Que se passe-t-il si on lance la commande `cat j` ?
7. Que se passe-t-il si on exécute `./c` ?
8. Écrivez une suite de commandes BASH qui, exécutées successivement dans un répertoire initialement vide quelconque, permet de rejoindre la même situation que celle donnée plus haut. [3]

Exercise 2 *Code de retour et séquences*

La variable `$?` contient le code de retour de la dernière commande exécutée au premier plan.

1. Que font les commandes `true` et `false` ?
2. En se basant sur l'aide et/ou le manuel de la commande `test`, dire comment tester
   a) qu'un fichier est présent ;
   b) qu'un fichier est présent et est un répertoire ;
   c) qu'un fichier est présent et est un fichier tube.
3. Comment est défini le code de retour d'un *pipeline* (*i.e.*, d'une séquence de commandes chaînées à l'aide d'un tube) ?
4. Comment peut-on connaître le code de retour de chacune des commandes impliquées dans un tube ? *(Indice : recherchez 'Paramètres spéciaux' et 'STATUS' dans le manuel de* BASH*.)*

1. Vous pouvez regrouper par type.
2. Doit-il être propriétaire ? membre du groupe propriétaire ? *etc.* . .
3. Vos commandes doivent fonctionner dans n'importe quel répertoire vide sur lequel vous avez les droits de lecture, d'écriture et d'*exécution*. Vous ne pouvez en particulier pas supposer l'existence de fichiers en dehors de ce répertoire, ni créer de tels fichiers.

Exercise 3 *Signaux*

Le but de cet exercice est de créer un script `race.sh` qui lance plusieurs lignes de commande en parallèle, et indique laquelle termine en premier. Les lignes de commande seront passées depuis l'entrée standard. Par exemple :

```
printf '%s\n' 'sleep 8; echo YES' 'sleep 3; echo NO' | ./race.sh
```

devra afficher, quelque chose du genre :

```
[111481] sleep 8; echo YES
[111482] sleep 3; echo NO
NO
-> [111482] a terminé en premier.
```

et **rien d'autre**.

1. Écrire un script qui, pour chaque ligne de commande lue depuis l'entrée standard, lance la ligne de commande en arrière plan, puis, une fois toutes les lignes de l'entrée standard traitées, se met en attente de la terminaison des sous-processus lancés.
2. En regardant les options de `wait`, faites en sorte que le script s'arrête dès qu'un des sous-processus à terminé.
3. Une fois le premier sous-processus terminé, forcer la terminaison des autres sous-processus. Faire en sorte que ce "nettoyage" n'engendre pas de messages d'erreurs.
4. Trouver comment déterminer la ligne de commande dont le processus a terminé en premier, et afficher un message comme sur l'exemple donné en début d'exercice.

L'ordre de lancement des sous-processus impacte grandement la compétition. Pour d'avantage de justice, on peut "endormir" les sous-processus à leur lancement, et les réveiller tous au moment du "top-départ".

5. Quelle option doit-on passer à `kill` pour lui indiquer d'endormir un (ou plusieurs) sous-processus ?
6. Quelle option doit-on lui passer pour indiquer de réveiller les sous-processus endormis ?
7. Compléter votre script.
8. À l'aide de variables tableaux, faire en sorte que le message annonçant le vainqueur de la compétition précise la ligne de commande correspondante, *e.g.*, dans l'exemple du début d'exercice, le message pourrait être :

   ```
   -> [111482] (sleep 3; echo NO) a terminé en premier.
   ```

Exercise 4 *Redirections*

1. La commande `tee` permet de dupliquer la sortie standard dans des fichiers. On veut définir un *wrapper* de cette commande, qui permette d'écrire également sur l'erreur standard.
   a) Définir une fonction `tee` qui appelle la commande `tee` avec tous les paramètres donnés.
   b) Modifier la définition de votre fonction pour que, lorsque le premier argument est `-e`, les lignes lues depuis l'entrée standard soient également écrites sur l'erreur standard.
2. Définir un script `surround.sh` qui prend en arguments deux chaînes de caractère `pre` et `post`, et affiche sur l'entrée standard les lignes lues depuis l'entrée standard préfixées par `pre` et suffixées par `post`. Par exemple :

   ```
   seq 4 | ./surround.sh '>>' '<<'
   ```

   devra produire sur la sortie standard :

   ```
   >>1<<
   >>2<<
   >>3<<
   >>4<<
   ```

3. Améliorer votre script, pour que des options `-f`, `-F`, `-l` et `-L`, chacune prenant un argument, permettent de définir un préfixe et un suffixe différent pour la première (***first***) ligne ainsi que pour la dernière (***last***). Par exemple, on pourrait vouloir appeler `./surround.sh -f '>>>' -L '<<<' '...' '...'`.