Introduction à Linux (IM713)

TP 2 – Manipulation de processus et du shell

Exercice 1 – *Lister les processus :* `ps`.

1. Lancez quelques programmes (navigateur,..etc) depuis l'interface graphique, 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).
   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. Sans fermer les programmes que vous venez de lancer, affichez la liste des processus associés à votre terminal. Affichez la liste des processus dont vous êtes propriétaire. À quoi correspondent les colonnes PID et PPID ?
4. La commande ps | wc compte deux processus en plus de ceux qui existent réellement lorsqu'on lance la commande. Pourquoi ?
5. Depuis le terminal, pressez la combinaison de touche `Ctrl-Z`). Quel est le résultat ? Pouvez-vous encore utiliser Kwrite ?

   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).
6. Exécutez `xclock &` puis `xemacs &` puis `jobs`. Que constatez-vous sur les numéros de job associés aux programmes ?
7. Lancez la commande `top`. Dans quel ordre les processus affichés sont-ils classés ?
8. 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. Explorez la page de manuel et testez certaines de ces options.

La commande `Ctrl-Z` (^Z) correspond à l'envoi d'un signal via la commande `kill`.
On dispose des commandes fg et bg. fg permet de poursuivre le déroulement du programme. bg le fait aussi mais il relance le programme en tâche de fond.

Exercice 2 – *Signaux.*

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.

1. 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`.

2. Lancez une commande en arrière plan. Quel est le comportement du processus associé lorsqu'il reçoit les signaux suivants :
— sigkill (9)
— sigstop (23)
— sigcont (25)

1 Programmation shell

1.1 Comparaisons d'entiers :

1. -eq : est égal à
   if ["a" − eq"b"]
2. -ne : n'est pas égal à
   if ["a" − ne"b"]
3. -gt : est supérieur à
   if ["a" − gt"b"]
4. -ge : est supérieur ou égal à
   if ["a" − ge"b"]
5. -lt : est inférieur à
   if ["a" − lt"b"]
6. -le : est inférieur ou égal à
   if ["a" − le"b"]
7. < : est inférieur à (entre doubles parenthèses)
   (("a" < "b"))
8. <= : est inférieur ou égal à (entre doubles parenthèses)
   (("a" < "b"))
9. > : est supérieur à (dans une expression entre doubles-parenthèses)
   (("a" > "b"))
10. >= : est supérieur ou égal à (dans une expression entre doubles-parenthèses)
    (("a" >= "b"))

1.2 Conditions :

```
mavariable=$1
if [ $mavariable -gt 12 ]
then
  echo "Error"
```

```
elif [ $mavariable -lt 5 ]
then
  echo "Perfect"
else
       echo "Ok"
 fi
```

1.3 Boucles :

```
while [ -z $reponse ] || [ $reponse != 'bonjour' ]
do
    read -p 'Dites "bonjour" : ' reponse
done

for nom in "Chahir" "Fougeray"
do
  echo "Bonjour $nom"
done

personnes="Chahir Fougeray"
for nom in $personnes
do
  echo "Bonjour $nom"
done

for i in {1..10}
do
    echo "$i"
done


k=0
for((i=1;i<=10;i++))
{
  k=$((k + 1))
}

echo $k
```

1.4 Fonctions :

```
function bonjour {
       echo "Bonjour $(whoami)
}
bonjour

function carre {
       echo $(($1*$1))
}
resultat==$(carre 3)
```

```
echo "Le carre de 3 est $resultat"
```

Exercice 3 – *Scripts shell*

1. Écrire un programme shell qui affiche les arguments du programme, dans l'ordre d'apparition (1er argument en premier). Si le programme n'a aucun argument, afficher « sans argument ».
   Indication : Utilisez la commande `shift`.
2. Créez un fichier exécutable, que vous nommerez `env_perso`, et que vous rendrez exécutable depuis n'importe quel répertoire de votre arborescence. Le résultat de cette nouvelle commande sera de vous afficher dans le terminal et de la façon la plus lisible possible :
   1. votre login
   2. la valeur de votre variable `HOME`
   3. la date du jour
3. Écrire un programme shell qui affiche tous les sous-répertoires du répertoire courant, en utilisant une `boucle`.
4. Proposez un programme shell qui qui calcule la factorielle de n.

Exercice 4 – *Les conditionnelles imbriquées*

1. Écrire un programme shell qui accepte 2 paramètres. Le premier paramètre est +r, -r, +w ou -w, et le deuxième paramètre spécifie une extension de nom de fichiers. En fonction de la valeur du premier paramètre, le programme modifiera les droits du groupe de tous les fichiers du répertoire courant dont l'extension est égale au deuxième paramètre. Pour contrôle, avant chaque modification des droits sur un fichier, le programme affichera le nom du fichier.
   Exemple d'utilisation : (le script s'appelle droitsfichiers)

   ```
   droitsfichiers +r .html
   ```

2. Proposez une nouvelle version de ce programme capable d'accepter trois paramètres. Les trois paramètres spécifient alors :
   — le répertoire dans lequel sont contenus les fichiers dont les droits seront modifiés,
   — les modifications des droits pour le groupe,
   — l'extension des fichiers concernés.
   Exemple d'utilisation : (le script s'appelle droitsfichiers)

   ```
   droitsfichiers .. -w .dat
   droitsfichiers perso -r .txt
   ```