ISS - Initiation aux Systèmes d'exploitation et au Shell
LU2IN020

TP 02 – Introduction à la notion de processus

Julien Sopena

septembre 2022

Le but de cette deuxième semaine est d'appréhender la notion de processus, d'étudier le lancement d'un programme depuis un terminal, d'apprendre à travailler avec la valeur retournée par le programme et de comprendre qu'une commande *Bash* n'est rien d'autre qu'un programme qu'on lance.

Exercice 1 : Question de nom

Question 1

Pour commencer nous allons récupérer les données nécessaires à l'ensemble des exercices. Placez-vous dans le répertoire correspondant à ce TP pour y extraire l'archive après l'avoir chargée à l'URL suivante : http ://julien.sopena.fr/LU2IN020-TP_02.tgz

Question 2

Après vous être déplacé dans le répertoire `exo1`, regardez le code `mon_test.c` et compilez-le avec la commande fournie dans le fichier README. Pour rappel, vous pouvez afficher le contenu d'un fichier avec la commande `cat`.

Question 3

Testez le fonctionnement de code avec différents arguments et en absence d'arguments.

Question 4

Pour plus de lisibilité, on veut renommer l'exécutable en `est_il_pair`. Pour cela vous pouvez utiliser la commande `mv`, déjà vu dans le TP précédent et qui permet de déplacer et/ou de renommer un fichier.

Une fois que cela est fait, relancez un test sans passer d'arguments. Le résultat est-il convenable ?

Question 5

Proposez une correction qui permette un affichage du message d'erreur correct, quel que soit le nom de l'exécutable. Testez votre solution.

Exercice 2 : Et PATH le chemin

Question 1

Avant de commencer ce deuxième exercice, vous allez devoir copier, à l'aide de la commande `cp`, une version de l'exécutable `est_il_pair` dans le répertoire `exo2`. Vérifiez que cela ai bien été réalisé en lançant la commande suivante :

```
moi@pc tp2/exo2 $ ./est_il_pair 24
```

Question 2

Essayez maintenant la commande suivante. D'où vient le problème ?

```
moi@pc tp2/exo2 $ est_il_pair 24
```

Question 3

Sauvegarder la variable `$PATH` dans une variable `$PATH_OLD`, puis corrigez le problème en ajoutant le répertoire `tp2/exo2` à la variable `$PATH`. Vous devriez pouvoir exécuter la commande précédente.

Question 4

Il peut être tentant d'ajouter `'.'` dans `$PATH` pour ne plus avoir de problèmes. Testez cette solution en exécutant la commande suivante.

```
moi@pc tp2/exo2 $ PATH=.:$PATH_OLD
```

Question 5

Avant d'aller plus loin, déplacez-vous dans le sous-répertoire `rep1` et créez-y un répertoire `rep2`.

Question 6

Pourquoi est-il dangereux de mettre `'.'` dans un PATH ?

Exercice 3 : Une histoire de famille

Question 1

Implémentez un script `fils.sh` qui affiche son *PID*, accessible depuis la variable `$$`, et le PID de son père, accessible depuis la variable `$PPID`, sous la forme du message suivant :

`"Je suis 234 et mon père est 127"`

Question 2

Implémentez maintenant un script `pere.sh` qui, après avoir affiché son PID, lance dix exécutions de votre script `fils.sh`.

Question 3

Pour simplifier le lancement des scripts, les systèmes Unix utilisent un nombre magique qui permet d'identifier un exécutable comme script, lorsqu'il est placé au début du fichier. Ainsi, lorsqu'un fichier exécutable commence par la séquence `#!`, appelé *shebang* suivi sur la même ligne du nom d'un interprète, le système sait qu'il ne s'agit pas d'un binaire. Il le fera alors exécuter par l'interprète indiqué.

Dans notre cas, pour chacun de vos scripts il vous faut :
— ajouter la ligne `#! /bin/bash` au début du fichier ;
— lancer la commande suivante pour rendre votre script exécutable

```
moi@pc /home/moi $ chmod u+x monScript.sh
```

Vous pouvez maintenant lancer tous vos scripts comme les autres exécutables :

```
moi@pc /home/moi $ ./monScript.sh
```

Corrigez vos scripts pour profiter de cette méthode. Est-ce que cela change quelque chose au nombre de processus créés.

Question 4

En dehors du *shebang* qui se trouve toujours au début du fichier, le caractère dièse `#` (en réalité le caractère *croisillon*) est utilisé comme caractère d'échappement pour les commentaires. Ainsi, pour chaque ligne de code tout ce qui se trouve après un `#` est ignoré.

Utilisez cette fonctionnalité pour commenter vos scripts `pere.sh` et `fils.sh`. Dans la suite de cette UE, nous vous demandons de bien commenter vos codes. Commentaires et indentation feront partie de l'évaluation de vos rendus.

Exercice 4 : Scripts paramétrés

Comme dans tous les langages, *Bash* offre une API pour accéder aux paramètres passés au moment du lancement de l'exécutable. Ainsi, il définit les variables suivantes :
— `$@` : liste de tous les paramètres
— `$#` : nombre de paramètres au sens strict, *i.e.*, sans prendre en compte le nom du programme.
— `$0` : nom du script
— `$1`, `$2`, `$3`, ... : respectivement premier, deuxième et troisième paramètre

Question 1

Pour commencer, copiez dans le répertoire `exo4` votre dernière version du script `pere.sh` et modifiez cette copie pour que le script affiche un message d'erreur puis s'arrête avec la commande `exit` s'il a été lancé sans paramètres.

```
moi@pc tp2/exo4 $ ./pere.sh
Il manque un paramètre
Usage : ./pere.sh <nb_fils>
```

Question 2

On veut maintenant utiliser ce paramètre pour configurer le nombre de fils créés. De plus, chaque fils devra afficher son rang en plus de son *PID* et de son *PPID*.

```
moi@pc tp2/exo4 $ ./pere.sh 5
Je suis 21304
Fils 1 : Je suis 21305 et mon père est 21304
Fils 2 : Je suis 21306 et mon père est 21304
Fils 3 : Je suis 21307 et mon père est 21304
Fils 4 : Je suis 21308 et mon père est 21304
Fils 5 : Je suis 21309 et mon père est 21304
```

Pour créer ces processus, vous pouvez utiliser la structure de contrôle **while** et un compteur. Comme pour le **if**, elle se base sur le code de retour de la commande qui suit le mot clé et répétera la boucle *tant que* cette commande retourne 0. On peut bien sûr utiliser la commande **test**, et son alias **'['**.

```
while commande ; do
   ...
done
```

```
while [ .... ] ; do
   ...
done
```