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

TD 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 : Code source, exécutable et processus

Dans ce premier exercice, nous allons étudier le code C d'un petit programme et son lancement par l'interprète du *Bash*. Ainsi on suppose qu'il existe, dans le répertoire courant, un fichier `sum.c` contenant le code suivant :

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char const *argv[])
{

        if (argc != 3) {
                printf("Il faut deux arguments\n");
                exit(-1);
        }

        printf("Le  résultat est : %d\n", atoi(argv[1]) + atoi(argv[2]));

        return 0;
}
```

Question 1

Pour commencer, étudiez la fonction `main()`. Que fait ce programme ? Quel affichage est produit, si l'on appelle le programme avec les paramètres 22 et 34 ?

Question 2

Le programme a besoin de deux paramètres, mais on fait le test sur `argc` par rapport à 3. Pourquoi ?

Question 3

Le terme programme est souvent employé à mauvais escient. On préférera donc les termes de code source, exécutable (ou binaire) et processus, moins ambigus. Illustrez ces trois termes avec notre cas d'étude (le cas échéant vous indiquerez la commande permettant de l'obtenir).

Exercice 2 : Le lancement d'un programme

Dans cet exercice nous allons étudier ce qui se passe lorsque nous lançons des programmes depuis la console. On supposera ici l'existence d'un binaire `sum` correspondant à la compilation du code étudié dans l'exercice 1.

Question 1

Pour commencer, rappelez les différentes étapes étudiées la semaine dernière de l'exécution de la commande suivante dans une console ; pour simplifier, on regroupera les substitutions au sein d'une même étape :

```
moi@pc /home/moi $ ./sum 23 12
```

Question 2

Quel lien existe entre les deux processus ?

Question 3

Comment le système d'exploitation identifie ces deux processus et leurs relations de filiation ?

Question 4

Dessinez le chronogramme correspondant à ce lancement.

Question 5

On considère maintenant le script `creation_dir.sh` dont le code est donné ci-dessous. Quelle commande permet de lancer ce script ? Combien de processus sont alors créés ?

```
mkdir LU2IN020
  for num in {0{1..9},10,11} ; do
  mkdir LU2IN020/TP_$num
done
```

Question 6

Peut-on générer la même arborescence en se limitant à la création d'un processus ? Quel en serait l'intérêt ?

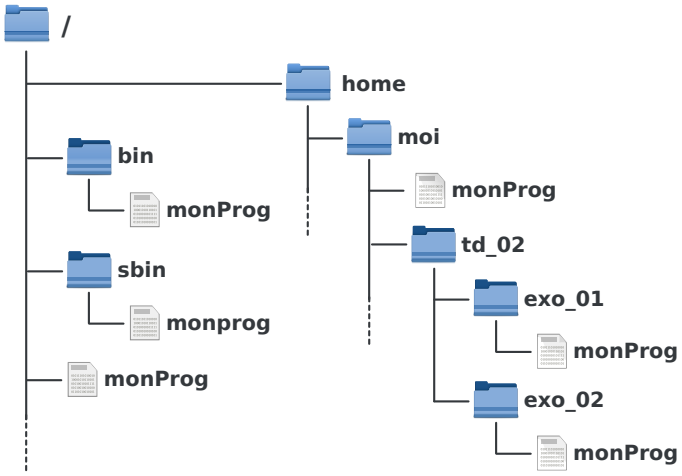
Question 7

On veut maintenant afficher tous les README présents dans les différents sujets de TP. Combien de processus sont créés par l'exécution du script suivant :

```
for num in {01..11} ; do
  cd LU2IN020/TP_$num
  pwd
  ls
  cat README
  cd ../..
done
```

Exercice 3 : Recherche de l'exécutable

Dans cet exercice, on considère l'arborescence suivante qui contient plusieurs instances d'un exécutable :



Question 1

On s'intéresse tout d'abord à la séquence d'instructions suivante :

```
moi@pc /home/moi $ echo $PATH
/sbin:/bin:/usr/bin:/usr/share/bin
moi@pc /home/moi $ pwd
/home/moi
moi@pc /home/moi $ monProg
```

Quelle instance de `monProg` va utiliser *Bash* pour lancer la commande, sachant que la recherche de l'exécutable se fait suivant l'algorithme suivant :

- si la chaîne est un chemin, i.e. contient au moins une fois le caractère `/`, *Bash* tente de l'exécuter ;
- si la chaîne n'est pas un chemin, bash vérifie si elle correspond à l'une des commandes *builtins* ;
- si ce n'est une commande *builtins*, *Bash* cherche itérativement dans tous les répertoires listés dans la variable `$PATH` sous la forme d'une suite de répertoires cibles séparés par le caractère `':'` ;
- en cas d'échec, *Bash* affiche un message d'erreur indiquant que l'exécutable n'a pas été trouvé.

Question 2

Proposez une ligne de commande qui modifie la variable `$PATH` de façon à ce que ce soit l'exécutable du répertoire `td_02/exo_01` qui soit lancé.

Question 3

En étant lancé depuis le répertoire `home` de l'utilisateur (`/home/moi`), le script suivant permet-il de lancer toutes les instances de `monProg` du *td_02*. Si oui pourquoi ? Si non, proposez une correction.

```
for dir in td_02/* ; do
  cd $dir
  monProg
  cd ../..
done
```

Exercice 4 : Test et exécution conditionnelle

Comme tous les langages le *Bash* offre une structure de contrôle conditionnelle. Ainsi le mot clé `if` permet de tester si la commande qui suit s'est exécutée sans problème, *i.e.*, a retourné la valeur `0`.

Voici un exemple d'utilisation qui suppose l'existence dans le répertoire courant d'un exécutable `monProg` correspondant au code *C* suivant :

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char const *argv[])
{
        if (argc < 2) {
                printf("Il faut un argument\n");
                exit(-1);
        }

        return atoi(argv[1]) % 2;
}
```

```
for num in {1..100} ; do
  if monProg $num ; then
    echo -n "$num "
  fi
done
echo
```

Question 1

Si la syntaxe du `if` est correcte, une erreur s'est glissée dans ce script. Trouvez là et proposez une solution.

Question 2

Quel est l'affichage produit par la version corrigée du script ?

Question 3

Pourquoi est-il difficile d'utiliser notre programme C pour faire le test inverse, sans avoir à le modifier.

Question 4

Lorsqu'une commande se termine, le *Bash* affecte sa valeur de retour à la variable `$?`. Attention, dès qu'on lance une nouvelle commande cette valeur est perdue, puisqu'elle remplacée par la valeur de retour de la dernière commande lancée.

Pour tester la valeur de la variable `$?`, le *Bash* offre la commande `test` qui permet d'évaluer une expression booléenne et retourne 0 si son résultat est vrai. Grâce à elle on peut utiliser le `if` du *Bash* comme on le fait dans les autres langages.

Il existe un grand nombre de tests possible : sur les chaînes de caractères, sur les valeurs entières ou encore sur le système de fichiers. Leurs descriptions sont à lire sur `man test`.

Dans un premier temps, utiliser l'expression `-lt` (*less than*) pour afficher tous les nombres impairs compris entre 1 et 100, sans modifier le programme C. La syntaxe est la suivante :

`test x -lt y` $\longrightarrow$ retourne 0 si $x < y$

Question 5

Pour simplifier son accès aux programmeurs habitués aux parenthèses, *Bash* introduit un alias à la commande `test` sous la forme de la commande `[` qui s'utilise exactement comme `test` avec un dernier paramètre `]` pour faire beau.

```
test x -lt y   ⟺   [ x -lt y ]
```

Réimplémentez votre script avec cette nouvelle syntaxe.

