

# **ISS - Initiation aux Systèmes d'exploitation et au Shell**

## **LU2IN020**

### **TD 07 – Cycle de vie d'un processus et exécution concurrente**

**Julien Sopena**

**novembre 2021**

Le but de cette septième semaine est d'étudier la commutation de processus, ainsi que l'automate décrivant la vie d'un processus. On y apprendra à lancer des commandes en parallèle, en s'attachant à observer les commutations de processus. Enfin, elle sera l'occasion d'introduire les concepts d'ordonnancement *batch* et *temps partagé*.

## Exercice 1 : Sans attendre

Comme on l'a vu dans les exercices des semaines précédentes, par défaut les commandes sont lancées dans un processus dédié par le processus exécutant le *Bash*. Ce dernier attendant alors la fin de l'exécution de la commande avant de pouvoir en lancer une autre. Il est cependant possible de lancer la commande tout en continuant l'exécution en plaçant un caractère *&* après la commande et ses paramètres. À noter qu'un *&* agit comme un séparateur de commandes.

Pour commencer, on supposera dans l'ensemble de cet exercice que les programmes s'exécutent sur un multicœur comprenant assez d'unités de calcul pour que tous les processus lancés puissent s'exécuter en parallèle, *i.e.*, c'est-à-dire qu'ils peuvent s'exécuter au même instant dans des unités de calcul différentes.

### Question 1

Faites 3 chronogrammes illustrant les différences entre les trois utilisations de `monScript.sh` suivantes.

1. `./monScript.sh`
2. `source ./monScript.sh`
3. `./monScript.sh &`

### Question 2

On suppose maintenant qu'un répertoire contient trois scripts exécutables : `A.sh`, `B.sh` et `C.sh`. Combien de processus seront créés à l'exécution de la commande `./A.sh` si le code des scripts `A.sh` et `B.sh` sont les suivants :

**A.sh**

```
#!/bin/bash

./B.sh
./B.sh &
```

**B.sh**

```
#!/bin/bash

for i in {1..3} ; do
    ./C.sh
done

for i in {1..3} ; do
    ./C.sh &
done
```

### Question 3

En vous aidant d'un chronogramme, calculer le nombre maximum de processus pouvant s'exécuter au même instant lorsqu'on lance la commande `./A.sh`. On vous rappelle qu'on ne peut faire aucune hypothèse sur la durée d'exécution des différents scripts.

### Question 4

Proposez maintenant un script `famille_nombreuse.sh` dont l'une des exécutions puisse conduire 10 processus frères à s'exécuter au même instant.

### Question 5

Pour finir, implémentez un script `longue_lignee.sh` pour qu'une de ses exécutions puisse conduire 10 processus de générations successives (père, fils, petit-fils, ...) à s'exécuter au même instant.

## Exercice 2 : Le partage

Dans ce deuxième exercice, on considère un ordinateur muni d'un CPU mono cœur, *i.e.*, qu'à un instant donné seul un processus pourra s'exécuter.

### Question 1

Quel composant des systèmes d'exploitation permet de partager la seule unité de calcul disponible ?

### Question 2

La vie d'un processus est rythmée par 3 états principaux : *prêt*, *élu*, *bloqué*. Dessinez l'automate régissant ces trois états. Votre schéma devra faire apparaître les différences entre les modes d'ordonnancement *batch* et *temps partagé*.

### Question 3

Les scripts `./pere.sh` et `fil.sh` suivants utilisent la commande `sleep`. Cette commande bloque le processus pendant un nombre de secondes donné un paramètre. Elle déclenche donc une élection et par la même un changement de processus appelé : *commutation*.

En vous aidant d'un chronogramme, donnez l'affichage produit par l'exécution de la commande `./pere.sh` sachant que l'on suppose :

- un système en mode *batch* ;
- une absence d'autres processus *prêts* à s'exécuter ;
- le maintien du père dans le CPU lorsqu'il crée un fils (c'est le comportement par défaut dans Linux ; nous reviendrons dessus en cours).

## pere.sh

```
#!/bin/bash

echo Debut pere

./fils.sh &

for i in 1 2 ; do
    echo Pere : $i
    sleep 1
done

echo Fin pere
```

## fils.sh

```
#!/bin/bash

echo Debut fils

for i in {a..c} ; do
    echo fils $i
    sleep 1
done

echo Fin fils
```