

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

## LU2IN020

### TP 08 – Parallélisation et synchronisation avec wait

Julien Sopena

novembre 2022

Le but de cette huitième semaine est d'étudier la parallélisation de commandes et la synchronisation à l'aide de la commande `wait`. Nous y introduirons aussi la notion d'adoption et de processus *zombie*.

#### Exercice 1 : AdopteUnZombie.com

Dans cet exercice on va utiliser la commande `ps` (déjà vu en TP) qui permet d'afficher des informations sur les processus présents dans le système. Nous utiliserons l'option `--pid` qui permet de cibler un processus en particulier, ainsi que la sélection des informations avec l'option de formatage `o`.

```
moi@pc /home/moi $ ps o pid,ppid,state --pid <UN_PID>
```

#### Question 1

Pour commencer, lancez la commande `sleep 30 &` et observez les informations sur le processus. Notons qu'avec ce qui a été vu en TD, il n'est pas nécessaire dans cette première question de recopier le *pid* du `sleep`.

#### Question 2

Implémentez maintenant un script qui permette ensuite d'observer (toujours avec la commande `ps`) l'adoption d'un processus par *init*.

#### Question 3

Observer un processus zombie en *Bash* est beaucoup plus compliqué. Pour commencer, implémentez un script `processMonitor.sh` qui lance toute les secondes pendant une minute, la commande `ps` sur le *pid* passé en paramètre.

Utilisez ce script pour surveiller le futur zombie, créer par le lancement de l'exécutable `myZombie` que vous trouverez dans l'archive <http://julien.sopena.fr/myZombie.tgz>.

#### Question 4

Cette question est à réaliser en dernier après avoir fini tous les autres exercices. Pour finir, modifiez votre script `processMonitor.sh` pour qu'il surveille sans rien afficher le processus dont le *pid* est passé en paramètre, jusqu'à ce qu'il devienne zombie ou qu'il disparaisse. Votre nouveau script `zombieDetector.sh` affichera alors un message d'alerte : "Le processus est devenu zombie !!!", puis s'arrêtera.

## Exercice 2 : D'un cœur à l'autre

Dans cet exercice, nous allons construire les fichiers nécessaires à l'exercice suivant. Vous devez donc le terminer avant de vous attaquer à l'exercice 3.

### Question 1

Pour commencer, télécharger le dictionnaire `dico.txt` à l'URL suivante :<http://lien.sopena.fr/dico.txt>

### Question 2

Après avoir créé un répertoire `dico`, générer un fichier par lettre contenant tous les mots commençant par cette lettre. Tous les mots du fichier `dico.txt` que vous avez téléchargé doivent donc être répartis dans les différents fichiers.

## Exercice 3 : À plusieurs

Dans cet exercice, on veut chercher le mot le plus long d'un dictionnaire. L'idée est d'accélérer cette recherche en lançant en parallèle la recherche des plus longs mots par initiale.

### Question 1

Pour commencer, implémentez un script `longest.sh` qui prend en paramètre le nom d'un fichier contenant une liste de mots (un mot par ligne) et qui va y chercher le mot le plus long. Une fois trouvé, votre script devra écrire ce mot dans un nouveau fichier dont le nom est le nom du fichier source suffixé par `.tmp` (*e.g.* le mot le plus long contenu dans `A.txt` sera écrit dans `A.txt.tmp`).

Votre script devra en outre vérifier la présence du paramètre et le fait qu'il s'agisse bien d'un fichier lisible. Si tel n'était pas le cas, il devra afficher un message d'erreur avant de se terminer.

### Question 2

Utilisez, sans le modifier, votre premier script pour implémenter un script `paraLongest.sh` qui affiche le mot le plus long contenu dans tous les fichiers d'un répertoire dont le nom sera passé en paramètre. Votre script cherchera à paralléliser au maximum cette recherche.

Comme dans la question précédente, vous vérifierez la présence d'un nom de répertoire comme paramètre.

### Question 3

Pour comparer les performances de vos deux scripts. Vous allez utiliser la commande `time`, en lançant `longest.sh` sur le fichier complet `dico.txt` et `paraLongest.sh` sur le répertoire `dico`. La commande `time` permet de mesurer le temps réel d'exécution d'une commande (temps mis pour obtenir le résultat), ainsi que les temps d'utilisation du processeur en mode user et en système (temps passé dans des appels système). Elle s'utilise de la manière suivante :

```
moi@pc /home/moi $ time cmd param ...
```

```
real 02m3,045s
user 1m2,020s
sys 1m1,025s
```



**Question 4**

Dans les mesures de temps obtenues, faites la somme des temps `user` et `sys` puis comparez-la au temps `real`, pour les deux scripts. Comment expliquer ce résultat ?

**Question 5**

Proposez une mesure du taux de parallélisme. Ce résultat vous paraît-il satisfaisant ?