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

TD 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 : Exploiter au mieux ses ressources

Dans cet exercice, on veut charger le film *la classe américaine* sur la plateforme de téléchargement légal *https ://mesfilmscultes.com*. Pour accélérer le chargement, les films y sont découpés en 100 morceaux et chacun de ces morceaux est compressé avec la commande `zip`. Ces fichiers sont téléchargeables à la racine du site sous le nom `nom_film-id.zip` où `id` est le numéro du morceau.

Question 1

Pour commencer, écrivez un script qui charge, puis décompresse et enfin réassemble les morceaux, pour obtenir `la_classe_americaine.avi`.

Question 2

Notre machine étant équipée de nombreux cœurs de calcul, on désire paralléliser certains traitements. Quelles étapes, nécessaires à la récupération du film, gagneraient à être parallélisé, *i.e.*, à être exécuté par plusieurs processus en même temps ?

Question 3

Modifier votre première version, en lançant certains traitements en parallèle de façon à exploiter au mieux les ressources disponibles.

Question 4

Cette dernière solution n'est pas fonctionnelle. En effet, elle suppose que les décompressions (lancées en parallèle) soient terminées lorsque le processus principal commence le réassemblement.

Corrigez ce problème en utilisant la commande `wait` qui bloque le processus qui l'utilise jusqu'à ce que *tous* ses fils soient terminés.

Question 5

Quel serait l'effet de l'ajout d'un `wait` juste après le lancement d'un `unzip` ?

Question 6

Votre dernière version du script est elle optimale en termes de parallélisation ?

Question 7

Proposez une nouvelle version plus efficace, sachant que :
— la variable `$!` contient le pid du dernier processus lancé en concurrence en utilisant un `&` ;
— la commande `wait` permet d'attendre la fin d'un processus précis si un `pid` lui est passé en paramètre.

Question 8

Au début de cet exercice, nous avons supposé un environnement d'exécution muni d'un multicœur et donc de pouvoir exécuter en parallèle plusieurs processus. Votre version parallèle du script de téléchargement, aurait-elle un intérêt sur une machine monocœur, *i.e.*, où les processus ne s'exécuteraient qu'en concurrence : un seul des processus prêts s'exécute à un instant donné.

Exercice 2 : Zombieland

Question 1

En supposant qu'il n'y ait pas d'autres processus, implémentez un script `pere_1.sh` et `fils_1.sh`, tel que le père fasse un appel à wait réellement bloquant. Puis, dessinez le chronogramme de ce scénario.

Question 2

On veut maintenant deux scripts, `pere_2.sh` et `fils_2.sh`, où l'appel à la commande `wait` est non bloquant (le fils est déjà terminé). Comme pour la première question, dessinez un chronogramme de ce nouveau scénario.

Question 3

Implémentez une troisième version de ces scripts, `pere_3.sh` et `fils_3.sh`, tel que le père ne fasse pas appel à la commande `wait` et se termine après son fils. Une nouvelle fois vous ferez un chronogramme.

Question 4

Pour terminer, faites une dernière version (`pere_4.sh` et `fils_4.sh`) et le chronogramme associé, dans laquelle le père se termine avant le fils sans avoir fait appel à la commande `wait`.

Question 5

Complétez l'automate décrivant la vie d'un processus, avec ce que vous venez d'apprendre dans cet exercice.

Exercice 3 : Le patriarche

Ce dernier exercice, qui ressemble beaucoup à un exercice d'examen, est à préparer chez vous pour la semaine prochaine. Il sera corrigé rapidement au début du TD. Pour en profiter, il faut avoir essayé de le faire seul.

Dans cet exercice, on considère les scripts suivants :

patriarche.sh

```
#! /bin/bash

echo "Je suis le patriarche"

./descendant.sh 3 &

echo "C'est la fin du patriarche"
```

descendant.sh

```
#! /bin/bash

i=$1
echo "Je suis un descendant $i"

while [ $i -gt 0 ] ; do
  i=$((i-1))
  ./descendant.sh $i &
done

echo "C'est la fin d'un descendant"
```

Question 1

Dessinez un chronogramme correspondant à l'exécution du script `patriarche.sh`, en supposant avoir assez de cœurs pour que tous les processus puissent s'exécuter en parallèle.

Question 2

Peut-on être certain que l'affichage de fin du patriarche se fasse après tous les autres ? Si oui pourquoi, sinon que faudrait-il faire pour s'en assurer ?