TP 4 - Système d'exploitation

ZZ1

L'objectif de ce TP est d'utiliser certains mécanismes de communication inter-processus. Ceci sera l'occasion de revoir les redirections et de manipuler les arguments de scripts BASH.

Les fichiers à télécharger sont disponible sur Moodle. Vous pouvez éventuellement téléchargez le script `set_font.sh`, afin d'utiliser la fonction `set_font` qui insère des séquences d'échappement pour la coloration (et autres altérations stylistiques) dans la sortie standard. Essayez par exemple :

```
$ source set_font.sh
$ echo "$(set_font red bold)rouge et gras$(set_font)"
```

(l'appel `$(set_font)` final permet de rétablir le style par défaut). Vous pouvez inclure `set_font.sh` dans vos script à l'aide de la commande **`source`**.

1 Producteur – Consommateur

Téléchargez les deux scripts `producteur.sh` et `consommateur.sh` et rendez-les exécutables. Le premier script créé un certain nombre (au plus 8) de *tâches* (représentées par des entiers) qu'il donne en sortie standard, à intervalles irréguliers. (Il affiche également quelques messages sur l'erreur standard.) Le second script traite toutes les tâches (représentées par des entiers) lues depuis l'entrée standard (une tâche par ligne), ce qui lui prend un certain temps (variable). Le traitement est représenté par des messages affichés sur la sortie standard. Vous ne devez pas modifier ces deux scripts.

▷ `mkfifo`, `&`, **`wait`**, `[`, **`test`** et **`sleep`**

1. Écrire un script `prodcons.sh` qui lance 4 producteurs (*i.e.*, 4 processus exécutant `producteur.sh`) et 8 consommateurs (*i.e.*, 8 processus exécutant `consommateur.sh`) en arrière plan, puis attend que tout le monde termine son travail. Les tâches produites par les producteurs doivent toutes être écrite dans un même fichier tube, qui sera lu par tous les consommateurs. Ainsi, chaque tâche produite sera traité par un consommateur.
2. Modifiez votre script pour que tous les messages (pas les tâches envoyées dans le tube) soient affichés sur la sortie standard du script.
3. Modifiez votre script afin que le nombre de producteurs à lancer soit donné en premier argument, et le nombre de consommateurs à lancer soit donné en second argument.
4. Modifiez votre script pour qu'il accepte un ou deux arguments. Avec deux arguments, il devra fonctionner comme à la question précédente. Avec un argument (de valeur n), il devra lancer 1 producteur et n consommateurs. Avec un nombre différent d'arguments, il devra produire un message d'erreur et terminer immédiatement avec le code d'erreur 1.

2 Lecteurs – Écrivains

Téléchargez le script `lecteur.sh` qui affiche la première et la dernière ligne d'un fichier, dont le chemin est donné en unique argument, séparées par la ligne "...".

▷ **read**, **head**, **tail**, **sed**, **cat**, **mv**, **mktemp**, **sleep**, **echo** et **flock**

5. Écrire un script `ecrivain.sh` qui modifie un fichier en déplaçant sa première ligne en fin de fichier. Le chemin vers le fichier sera donné en unique argument du script. Par exemple, si le fichier `noms.txt` contient :

```
Joséphine Carlier
Roland Tessier de la Lejeune
Guillaume Blin
Laurent Guillon
```

son contenu sera le suivant, après l'appel de `./ecrivain.sh noms.txt` :

```
Roland Tessier de la Lejeune
Guillaume Blin
Laurent Guillon
Joséphine Carlier
```

Pour réaliser cela, vous pourrez utiliser un fichier temporaire (*c.f.* **mktemp**).

6. Les deux scripts `ecrivain.sh` et `lecteur.sh` peuvent-ils être en concurrence ? Y-a-t-il une ressource critique ? Si oui, réalisez un scénario montrant un résultat incohérent (pour cela, vous pouvez tricher, en ralentissant l'exécution des opérations dans les deux scripts — insérez des **sleep** `.25` adéquates, par exemple).
7. Écrire un script `lecteur-ecrivain.sh` qui lance, en arrière plan, e écrivains et l lecteurs, tous appelés sur un même fichier f où e, l et f sont respectivement donnés en premier, deuxième et troisième argument du script. Une fois lancé, le script attendra que tous (lecteurs et écrivains) finissent. Testez. Constatez-vous des résultats incohérents ?
8. Rétablissez la cohérence à coup sûr, en utilisant l'outil de verrouillage à double phase (*c.f.* cours) **flock** (*c.f.* **man flock** aussi). Avec quelle option doit-il être appelé dans `lecteur.sh` ? Dans `ecrivain.sh` ?
9. À l'aide de la commande **sleep** `.$RANDOM`, qui force une pause d'une durée aléatoire inférieure à 1 seconde, différez aléatoirement le lancement des processus lecteurs et écrivains dans `lecteur-ecrivain.sh`. Identifiez également chacun par un numéro [1], et faites-en sorte qu'il affiche des messages concernant les opérations qu'ils réalisent sur l'erreur standard. En particulier, on voudra savoir :
   — quand un verrou est demandé/obtenu/libéré ;
   — quel type de verrou est demandé/obtenu/libéré, par qui.

3 Signaux

▷ **trap**, **kill**, **sleep** et **wait**

10. Écrire un script `ilestquelleheure.sh` qui affiche l'heure (à la seconde près) toutes les 5 secondes (dans une boucle infinie), après avoir affiché son **pid**.
11. Exécutez le script et envoyez-lui le signal `SIGUSR1`. Que se passe-t-il ? Tuez le processus s'il est encore en vie.
12. En utilisant la commande **trap**, modifiez votre script pour qu'à la réception du signal `SIGUSR1`, il affiche l'heure (à la seconde près) dans un message de la forme : `"Vous demandez l'heure ? La voici : 09:45:38"`. Le script devra ensuite se remettre en attente. Testez. L'effet est-il immédiat ?

1. Vous pouvez ajouter un second argument aux scripts `lecteur.sh` et `ecrivain.sh` afin de leur passer un identifiant déterminé par `lecteur-ecrivain.sh`.

13. La commande `sleep` masque les signaux tant qu'elle s'exécute, d'où un certain délai (d'au plus 5 secondes) qui rend l'envoi du signal tout à fait inintéressant. Fort heureusement, la commande `wait`, elle, ne masque pas les signaux. Modifiez votre script, afin que l'effet de `SIGUSR1` soit immédiat.

14. Interceptez le signal `SIGUSR2` afin que le processus redonne son **pid** avant de continuer (en reprenant le délai à 0)...

15. Interceptez le signal `SIGQUIT` (qui peut être envoyé au processus de premier plan via la combinaison `ctrl+\`) afin qu'un message indique à quelle heure le processus a terminé, avant de terminer.