Fabien CALCADO

TP 2

Processus

Le TP a pour but d'apprendre à manipuler les processus UNIX.

Question 1.1
Ecrire un programme permettant de créer deux processus. Vous définirez une variable globale dans le programme. Les processus père et fils devront modifier cette variable globale (par exemple globale+=10 pour l'un et globale*=2 pour l'autre). Les opérations de modification seront à effectuer à plusieurs reprises (bouclées). Une macro-constante pourra être définie pour changer aisément le nombre de tours de boucle à effectuer.

Le programme affichera dans un premier temps la valeur initiale de la variable globale avant la création des processus.

Chaque processus devra afficher, où *type* correspondra soit à *pere* soit à *fils* et *XXX à* la valeur du PID :
- son PID et son PPID ainsi que la valeur de la variable globale sous la forme :

`[type] PID=XXX, PPID=XXX, v_globale=XXX`

- Après chaque modification de la variable globale (dans la boucle), le processus affichera son PID entre chevron suivi du numéro d'itération ainsi que la (nouvelle) valeur de la variable globale sous la forme

`[type] <XXX> : i=XXX, v_globale=XXX`

Vous ferez également un affichage juste avant le "return 0" du main indiquant la fin des processus (avec son PID) ainsi que la valeur de la variable globale finale.

`** Fin du processus <XXX>, v_globale=XXX **`

Refaire l'exercice avec des threads, quelle différence y-a-t-il ? Expliquez en la raison.

Question 1.2
Si vous omettez les '\n' au niveau des affichages, que constatez vous lors de l'exécution ? Expliquez exactement ce qu'il se passe. Même en ayant mis tous vos '\n' il est fort probable que certains affichages se fasse après la récupération du prompt à la fin de l'exécution. Expliquez pourquoi puis proposer une solution pour l'éviter.

Question 1.3
A quoi correspond la valeur renvoyée par la primitive fork() ? A quoi correspond le PPID du père qui est affiché par votre programme ? Vérifier que c'est bien le cas en exécutant 2 commandes différentes sur le terminal.

Question 1.4
Insérer des temps d'attente (fonction sleep ou usleep) constant dans chacune des boucles de modification. Est ce que les résultats changent (valeurs finales de v_globale) ?

Question 1.5
Refaites l'exercice avec des temps d'attente aléatoire. Vous afficherez le temps d'attente généré à chaque tour de boucle. A priori, vous devriez avoir tous coder la chose d'une façon impliquant que les temps d'attente soit exactement les mêmes dans chaque processus. Si cela n'a pas été le cas, modifiez votre programme pour montrer ce cas. L'important étant de bien en comprendre la raison.
Comment procéder pour que les temps d'attente soit bien différent entre les processus ? Qu'engendrent ces temps d'attente aléatoire sur l'exécution globale du programme (i.e. des processus) ?

Question 2.1
Ecrire un programme permettant de créer deux processus (afficher les PID et PPID du père et du fils). Créez une attente pour le fils. Des affichages encadreront cette attente

```
[fils] Je m'endors pour XXXs...
[fils] Je suis reveille !
```

Vous devrez synchroniser le père sur le réveille de son fils. Il attendra donc le réveil de son fils (i.e. fin d'exécution). Faire un affichage : `[pere] En attente de mon fils...` pour constater cette attente.

Comme dans la question 1.1 vous afficherez juste avant le" return 0" du main, où XXX est le PID du process :

```
** Fin du process <XXX> **
```

Question 2.2
Reprendre le programme de la question précédente. Insérer un appel système permettant de remplacer le code du processus fils par le code de la commande "echo" (ne pas utiliser la fonction *system( )* donc) avec comme paramètre "Bonjour" et un prenom stocké dans un tableau (saisie utilisateur). Ne pas modifier le reste du code. Que constatez-vous lors de l'exécution vis à vis du code suivant l'appel système ?

Question 2.3
Ecrire un programme "prog_fils" charger d'afficher "[fils]" suivi de son PID et PPID. Puis de la commande ayant permis de lancer son exécution ainsi qu'un "Bonjour" suivi du 1er paramètre de lancement. Tester votre programme. Avec la ligne de lancement "./prog_fils saymyname". Vous devriez obtenir à l'écran ceci :

```
[fils] PID <XXX> PPID <XXX>
[fils] commande de lancement : ./prog_fils
[fils] Bonjour saymyname
```

Question 2.4
Modifier le programme en 2.1 pour que le fils exécute le programme "prog_fils" et non "echo".

Question 2.5
Retester vos programmes précédent avec l'appel système execlp() au lieu d'execl(). Que se passe-t-il ? Insérer un appel système permettant de capter (simplement) l'erreur. Que faudrait-il faire pour que cela fonctionne ? Tester votre solution.
Note : n'utilisez pas d' "export" : vous pourrez ainsi vérifier que cela fonctionne dans le terminal courant et tester dans un autre terminal que l'erreur apparait toujours.

Question 2.6
Reprenez le programme 2.2 en créant un ensemble de processus fils. Ce nombre sera donné par l'utilisateur. Le père devra se réveiller lorsque tous ses processus fils seront terminés.