TP - ASR5 Système d'Exploitation

Variables d'environnement et processus

2018/02/06

I Utilisation des variables d'environnement

Le but de l'exercice est de manipuler des variables d'environnement, vous pouvez lister celles qui existent actuellement avec la commande env

Q.I.1) - En utilisant une variable d'envirronement, faites un programme qui affiche : « `Bonjour p10...` » où « `p10...` » est votre login.

Q.I.2) - `date` est une commande qui permet d'afficher la date courante. Sur les machines de l'université, l'affichage a été configuré pour être par défaut en français via la variable LANG.

```
mer. janv. 31 14:03:25 CET 2018
```

2(a) - Que ce passe-t-il si la variable LANG n'a pas de valeur ?

2(b) - Pouvez-vous afficher la date en allemand ou finnois ?

> **Solution:** Il faut modifier la variable `LANG` lors de l'appel. Pour cela, il suffit de faire précéder la commande par la valeur de la variable :
>
> ```
> ## pour avoir la valeur par défaut on supprime la valeur de
> ## la variable LANG
> LANG= date
> ## de façon étonnante, c'est de l'anglais !
> ## pour l'allemand ou le finois (testé sur les machines de
> ## la salle TP14)
> LANG=de_DE date
> LANG=fi_FI date
> ```
>
> J'ai trouvé la liste des langues possibles ici `https://www.w3.org/WAI/ER/IG/ert/iso639.htm` avec une recherche google de LANG fr_FR@utf8
>
> Toutes les valeurs de la norme ne sont pas possibles, cela doit dépendre de l'installation de certains packages, du travail de traduction fait par la communauté ...

Q.I.3) - Faire un programme qui affiche « `Bonjour p10...` » si la langue est française mais « `Hello p10...` » si elle ne l'est pas.

II Création de processus avec `fork()`

La fonction fork() permet de dupliquer un processus en créant un processus fils identique.

Q.II.1) - Consultez le manuel de la fonction fork() (utilisable en C ou C++). Quelles valeurs retourne t'elle ?

> **Solution:** La fonction retourne une valeur différente selon le processus et en fonction du succès ou non de son exécution.

— Si cela fonctionne, le fils reçoit 0 (cela permet de savoir qu'on est dans le processus fils) et le père reçoit un nombre strictement positif. Ce nombre est le numéro de processus du fils.
— Si `fork()` échoue, le résultat est -1 dans le père et il n'y a pas de fils.

Q.II.2) - En utilisant fork(), créez une petite famille de processus qui affichent chacun leur identité (`pid`). Vous devez faire un père et son fils.

Q.II.3) - Faites en sorte que les processus durent longtemps [1] et vérifiez leur « parenté » en affichant précisément la liste des processus (consultez les options de la commande `ps` ou de `pstree`).

Solution: ps auxf ou pstree. Il y a aussi un outil graphique pour voir les processus dont le nom dépend du gestionnaire de fenêtre utilisé. On le trouve généralement dans le groupe des applications système.

Q.II.4) - Le processus fils a-t-il conservé les variables initialisées par le processus père ? Démontrez le.

Solution: Oui, les variables sont conservées. Il suffit de créer des variables et de les faire afficher avant et après `fork()`. Par contre toute modification après le `fork` ne concerne que le processus qui fait la modification.

Q.II.5) - *Les processus orphelins* : si le père meurt avant son fils, celui-ci devient orphelin ; que se passe-t-il ?

5(a) - Recréez une descendance complète de processus (grand-père, père, fils). Ces processus doivent durer longtemps (sleep()).

5(b) - Que se passe-t-il si on coupe la branche de parenté au niveau du père ? Tuez le processus correspondant et vérifiez l'état des processus restants.

5(c) - Existe t'il un processus orphelin ? À quels processus a-t-il été rattaché ?

5(d) - Existe-t-il un processus zombi ? Lequel ?

5(e) - Pourquoi un processus sous Linux ne peut-il rester sans processus père ?

Solution: En résumé, lorsqu'un processus se termine, il devient `defunct` c'est-à-dire zombi. Il le reste tant que son père ne fait pas de waitpid(). Cela doit être le cas de votre processus père.
Le fils d'un processus mort (dont on sait que le père ne fera jamais de waitpid()) est *adopté* par un processus spécial qui est l'ancêtre de tous les processus de l'utilisateur (systemd −−user) ou l'ancêtre de tous les processus du système (systemd avec le pid 1). Le rôle de ce processus ancêtre est entre autre de *nettoyer* les processus zombies en adoptant les processus sans père et en faisant `wait()` automatiquement lorsqu'ils se terminent. Dans votre cas, le fils doit donc avoir été adopté, ce qu'on voit facilement avec `pstree`.
Le seul cas créant un zombi est celui ou un processus meurt et où son père ne fait pas de `waitpid` . Le système doit le conserver car il ne sais pas si un jour son résultat sera utilisé. À cause de cela, vous devriez voir `defunct` apparaître sur le processus que vous avez tué.

Annexe I

Comment gérer les processus sous Unix ?
Vous n'aurez besoin que des raccourcis clavier :

1. grâce à la fonction `sleep()`

— Ctl−c pour envoyer le signal INT à l'application courante (généralement elle se ferme) ;
— Ctl−\ pour envoyer le signal QUIT à l'application courante (généralement elle se ferme) ;
— Ctl−z pour envoyer le signal STOP à cette application (généralement elle s'arrête) ;
— ps pour afficher les processus en cours (voir les options a, x, l et f) ;
— kill pour envoyer un signal quelconque ;
— fg et bg pour passer une tâche stoppée en tâche de fond ou au premier plan.

Si toutes ces commandes vous rebuttent, essayer l'application graphique ksysguard.

Rappel : la syntaxe d'une commande peut s'obtenir avec man ou info.

Annexe II

Modifier le comportement de kdbg lors d'un fork

`kdbg` n'est qu'une interface graphique pour le logiciel en ligne de commande `gdb`. Comme toute inteface, il est pratique mais imparfait. La seule méthode pour modifier le comportement lors d'un fork est de donner l'ordre à `gdb`. Pour cela, il est possible de modifier la commade que `kdbg` utilise au lancement du programme :

— Ouvrez une première fois le programme par la commande `kdbg nomprog &`
— dans le menu `setting->this programme...` entrez la commande
`gdb --fullname -x gdb_fils.txt --nx`
— **Fermer Kdbg**
— copiez le fichier `gdb_fils.txt` sur le répertoire depuis lequel vous lancez `kdbg`
— Relancez `kdbg`, et vérifiez que la fonction `fork` renvoie 0.

III Utilisation des signaux

Vous devez ajouter un timer au programme signal_temps.cpp [2]

Ce dernier fait une boucle infinie et vous devez faire en sorte qu'il appelle une fois par seconde la fonction top. Pour cela vous utiliserez les fonctions :

— unsigned int alarm(unsigned int s); qui permet d'envoyer un signal au bout de s secondes
— sighandler_t signal(int signum, sighandler_t handler) qui permet de modifier la réaction usuelle à un signal.

2. .http://perso.univ-lyon1.fr/fabien.rico/site/_media/systeme:2018p:signal_temps.cpp