

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'environnement, 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 ?
- 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 ?
- 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<sup>1</sup> et vérifiez leur « parenté » en affichant précisément la liste des processus (consultez les options de la commande `ps` ou de `ptree`).
- Q.II.4)** - Le processus fils a-t-il conservé les variables initialisées par le processus père ? Démontrez le.
- 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 ?

---

1. grâce à la fonction `sleep()`

## Annexe I

*Comment gérer les processus sous Unix ?*

Vous n'aurez besoin que des raccourcis clavier :

- 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 interface, 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 commande 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`<sup>2</sup>

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