

TP - LIF12 système d'exploitation

Administration Système – GNU/Linux rules

2015/09/15

I Création de processus avec `fork()`

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

- Q.I.1)** - Consultez le manuel de la fonction `fork` (utilisable en C ou C++). Quelles valeurs retourne t'elle ?
- Q.I.2)** - En utilisant `fork()`, créez une petite famille (un père et son fils) de processus qui affichent chacun leur identité.
- Q.I.3)** - Vérifiez leur « parenté » en affichant précisément la liste des processus (consultez les options de la commande `ps`).
- Q.I.4)** - En utilisant le débogueur `kdbg`, suivez pas à pas le déroulement du programme. Lors du `fork()` que ce passe-t-il ? Quel processus suit le debugger ? Si vous avez le temps, faites en sorte de suivre l'autre.
- Q.I.5)** - Le processus fils a-t-il conservé les variables initialisées par le processus père ? Démontrez le.
- Q.I.6)** - Créer une descendance complète de processus avec trois niveaux de branches (grand-père, père, fils).
- Q.I.7)** - Les processus orphelins : si le père meurt avant son fils, celui-ci devient orphelin.
 - 7(a)** - Recréez une descendance complète de processus (grand-père, père, fils). Ces processus doivent durer longtemps (utilisez la fonction `sleep`). Que se passe-t-il si on coupe la branche au niveau du père ? Tuez le processus correspondant et vérifiez l'état des processus restants.
 - 7(b)** - Existe t'il un processus orphelins ? A quels processus a-t-ils été rattaché ?
 - 7(c)** - Existe-t-il un processus zombies ?
 - 7(d)** - Pourquoi un processus sous linux ne peut-il rester sans processus père ?

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-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.

II Utilisation des signaux

Vous devez ajouter un timer au programme `signal.temps.c`. 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.