Test de Connaissances de Systèmes d'Exploitation 2012

1 Questions

- À l'intérieur d'un programme, comment peut-on différencier entre le code d'un processus qui fait `fork()` et celui de son fils ?
- Quelles sont le fonctions système à l'aide desquelles un parent s'assure de la terminaison de ses fils ?
- Donnez la définition d'un zombie. Quand le système peut-il nettoyer les zombies créés par un processus négligeant ?
- Il y a une contrainte sur les processus pouvant communiquer par un pipe anonyme (`pipe()`) – laquelle ?

2 Mini-Exercices

- Écrivez l'appel système et les déclarations nécessaires pour bloquer le signal `SIGINT` pendant l'exécution d'un programme.
- Écrivez l'appel système et les déclarations nécessaires : changer vers `Derout` le traitant de signal pour le signal `SIGUSR1`. Pendant l'exécution du traitant `Derout`, bloquez le signal `SIGINT`.

3 Exercice

On veut écrire quelques composants principaux d'un programme nommé **batchserv** qui permet le lancement et suivi d'autres programmes, qu'on appellera ici **jobs**. Le suivi comprend ici la possibilité de le tuer. Un exemple de **session** suit :

```
allegro % ./batchserv.run
batch > charger job5.sh 4
Ok, chargement du job 0 -- ' job5.sh 4'
batch > executer 0
[Job 0] en_cours_d'execution
batch > charger job4.sh ccc 4
Ok, chargement du job 1 -- ' job4.sh ccc 4'
batch > executer 1 [Job 1] en_cours_d'execution
[Job 0] execution_complete sortie normale code 0 b
[Job 1] execution_complete sortie signal 15
batch > quitter
```

De plus, des fichiers sont également créés : un fichier de log par job, avec les sorties standard et erreur du job.

3.1 Structure générale

L'application est composée de plusieurs processus :

1. le processus père **dispatch**, qui affiche le prompt, assure le dialogue avec l'utilisateur, et se duplique lors des commandes "charger"
2. pour chaque commande "charger", un **escort** (fils de **dispatch**), qui
   - est lié par deux pipes anonymes avec son père le **dispatch** (pour l'écouter et lui répondre)
   - se duplique pour lancer avec `exec()` le **job**
   - attend des « événements » (**dispatch** lui parle, quelque chose arrive au **job**, etc.) et les « traite » en en communiquant sur un pipe le rapport.

3.2 Travail à faire

- Faites un schéma (avec des flêches) représentant les processus et leurs communications.
- Écrivez le code source en lignes générales, en laissant de l'espace pour les détails de chaque processus. Autrement dit : écrivez d'abord les instructions communes, à effectuer avant toute duplication (descripteurs de pipe, etc.). Écrivez ensuite les duplications (création des fils), et dans chaque processus mettez simplement des lettres `A`, `B`, `C`, `D` pour indiquer leur corps à écrire plus tard. N'oubliez pas de prendre soin de zombies et faites attention à ce qu'on ne fasse pas de « petits lapins » non plus.
- Uniquement une fois que l'architecture vous semble bien claire, écrivez la communication **escort-dispatch**
- Pour un bonus, écrivez au choix les détails de "charger" ou celles de "tuer".