TP 1 - 2: Fork

Exercice 1. Zombies

Ecrire un programme *zombie1.c* qui crée un processus fils par un `fork()` :

- le fils affiche son *ppid* et son *pid* et termine aussitôt.
- le processus père boucle (*i.e.* boucle infinie) sur un `sleep` de 10s.

Lancez le programme et observez l'état du fils :

```
% ps -o ppid,pid,state,cmd -p mettez_ici_le_pid_du_fils
```

- Tentez de tuer le fils par un `kill` (sous *Bash*). Conclusion ?
- Tuez le père par un `kill` et relancez la commande `ps` précédente. Conclusion ?

Ecrire un programme *zombie2.c* qui crée un processus fils par un `fork()` :

- le processus fils boucle (*i.e.* boucle infinie) sur un sleep de 1s et affiche avant chaque sleep le *pid* de son père.
- le père affiche le *pid* de son fils et termine aussitôt.

Observez le *pid* du fils. Tentez de le tuer par un `kill`. Conclusion ?

Exercice 2. Double fork

Si vous avez bien compris les deux programmes précédents, vous devriez pouvoir écrire un programme qui ne peut pas générer de zombies. L'idée à mettre en œuvre est simple :

> pour qu'un processus ne reste pas à l'état zombie, il faut être sûr que son père effectue un wait sur ce processus fils.

D'autre part :

> un processus dont le père est terminé est « adopté » par le *processus 1* (ou un processus dit "subreaper") qui garanti l'attente de terminaison du fils par un wait.

Les deux concepts mis bout à bout donnent lieu au concept de *double fork* :

au lieu d'exécuter le code directement dans le processus fils (fils1), celui-ci crée directement un autre fils (fils2) qui exécute le code puis termine normalement. Le premier fils (fils1), termine normalement et le processus père effectue un waitpid sur ce fils (fils1).

La terminaison du *fils1* provoque l'adoption du *fils2* par le *processus 1* qui effectue le `wait` sur le *fils2*.

Ecrivez un programme *double_fork.c* qui affiche son *pid* puis qui exécute le code suivant avec un double fork :

```
1 printf("Don't talk, think that's a good rule of thumb for life. (Rick
     Grimes)\n");
2
3 sleep(10);
```

Vérifiez que le processus est bien adopté par le *processus 1* avant de quitter.

Exercice 3. Exec

- Réalisez un programme qui en utilisant *execlp* réalise l'exécution de la commande `ps aux`.
- Même question avec *execvp*.
- Toujours la même question avec execl et execv. Pour utiliser ces fonctions, vous vous rappellerez que la commande `ps` se trouve dans le répertoire `/usr/bin`.