TD/TP 4

Processus, fork(), wait()

Exercice 1 (`fork()` mise en jambe)**.**
Qu'affiche le bout de programme suivant et dans quel ordre ?

```
...
n = 1;
printf("n = %d\n", n);
if (fork()) {
  printf("Je suis le père\n");
  n++;
} else {
  printf("Je suis le fils\n");
}
printf("maintenant n = %d\n", n);
...
```

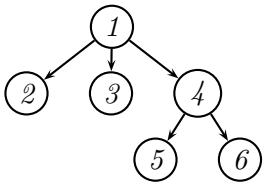
Et si on remplace le "`\n`*" par un espace, après* `père` *et après* `fils` *?*

Exercice 2 (`fork()`)**.**
Dessiner l'arbre des processus engendré par le programme suivant :

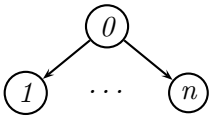
```
int main() {
  fork() || (fork() && fork());
  exit(EXIT_SUCCESS);
}
```

De même avec la ligne : `fork() && (fork() || fork());`*.*

Exercice 3 (`fork()`)**.**
Écrire un programme qui engendre l'arbre généalogique suivant :



Écrire un programme qui engendre l'arbre généalogique suivant :



Pour ce dernier on testera la bonne exécution des `fork()`*.*

Exercice 4 (Chasse aux zombies.)**.**

1. *Rappeler ce qu'est un processus zombie.*
2. *La balayette à zombies. L'appel* `waitpid(-1, NULL, WNOHANG)` *renvoie -1 en cas d'erreur. Si il n'y a pas de fils zombie il renvoie immédiatement 0 (l'appel est non bloquant) et sinon un fils zombie est choisi, toutes ses ressources sont libérées et son pid est renvoyé. En déduire une manière pour un processus de se débarrasser de tous les zombies qu'il a créé.*

Manipulation 1 (Ordonnancement avec usleep())**.**
En vous aidant de la commande `usleep()` *essayer d'obtenir toutes les sorties prévues en TD pour le programme de l'exercice 1.*

Manipulation 2 (fork() et ps)**.**

1. *À l'aide du man de la commande* `ps` *trouver une commande permettant d'afficher l'arbre des processus issue d'un processus initial.*
2. *Écrire une fonction C qui affiche l'arbre des processus du processus courant (utiliser* `system`*).*
3. *Tester votre fonction sur les programmes écrits en TD.*