TD/TP de Système n° 9 : `fork - exec - wait`

Exercice 1 :

1. Sans utiliser la fonction `system`, écrire un programme qui lance la commande passée en argument (avec ses options). Par exemple, `lance ls -l` exécute `ls -l`.
2. Modifier ce programme pour qu'il lance la commande dans un processus fils. Le père devra afficher le message `exécution réussie` après la mort de son fils.
3. On suppose écrits des programmes `plus`, `moins`, `fois`, `div`, `mod` prenant deux arguments (supposés entiers) et affichant le résultat (entier) de l'opération correspondante si elle est valide. Ces programmes retournent 0 en cas de succès, 1 en cas d'échec (en particulier en cas de division par 0).

   Écrire un programme `calcule` prenant trois arguments formant une opération arithmétique (par exemple `calcule 3 + 4`), lance un fils chargé d'effectuer l'opération, et affiche éventuellement un message d'erreur si l'opération a échoué.

Exercice 2 :

Que fait le programme suivant ?

```
#define N 10
int main() {
  int i = 1;
  while (fork() == 0 && i <= N) i++;
  printf("%d\n",i);
  exit(0);
}
```

Exercice 3 : `fork`–bombe

Combien de processus le programme suivant crée-t-il ?

```
int main(int argc, char *argv[]) {
  fork();
  fork();
  fork();
  exit(0);
}
```

Dessiner l'arbre des processus engendrés. Remplacez le premier appel à `fork()` par un appel à `execvp` qui exécute le programme lui-même. Que se passe-t-il ? Que se passe-t-il si on remplace le deuxième ?

Julien Cristau, Anne Micheli, Samuel Mimram

Exercice 4 : `forkn`

Écrire une fonction `forkn(int n, void (**tab)())` prenant en argument un entier n et un tableau de n pointeurs de fonctions, et lançant n processus fils ; le i^{e} fils doit exécuter la i^{e} fonction avant de terminer. Comment faire pour que les fils s'exécutent en parallèle ? et l'un après l'autre ?

Exercice 5 : recherche parallélisée dans un tableau

On souhaite trouver toutes les occurrences d'un élément dans un tableau donné. Pour cela, si le tableau est long (plus qu'une constante `TAILLE_MIN`), on peut le couper en deux et effectuer la recherche indépendamment dans les deux moitiés, en les confiant à deux processus différents travaillant en parallèle.

Écrire un programme effectuant la recherche de cette manière ; le programme devra afficher un message pour chaque occurrence trouvée, puis afficher le nombre total d'occurrences de l'élément dans le tableau. Pour cela, on pourra utiliser les valeurs de retour des processus fils (ces valeurs de retour sont codées sur un octet).

Exercice 6 : processus en cascade

Écrire deux programmes qui créent n processus p_i tels que :

1. pour tout i entre 2 et n, p_i soit le fils de p_{i-1} ;
2. pour tout i entre 2 et n, p_i soit le fils de p_1.

Chaque processus devra afficher son identifiant et celui de son père.

Exercice 7 :

Écrire deux programmes en `C` correspondant aux commandes shell suivantes :

1. `gzip toto && echo fichier zippé`
2. `gzip toto || echo problème`

Exercice 8 : `find` parallélisé

Écrire une commande `find_parallele` telle que `find_parallele` *fic rep* effectue la recherche d'un fichier de nom *fic* dans le répertoire *rep*. Pour cela, chaque processus parcourt le contenu du répertoire dont la référence lui est fournie ; s'il trouve un fichier de nom *fic*, il en affiche la référence complète ; et pour chaque sous-répertoire, il crée un processus fils chargé d'y poursuivre la recherche.