Sujet du TP n° 4

Préambule

Les ressources du cours (supports, exemples, et autres documents) et des TP (les sujets, les fichiers de travail ainsi que les corrigés) se trouvent à l'URL : http://www.emse.fr/~lalevee/ismin/pse, dénommé **[site]** dans les documents fournis.

Ce TP suppose que vous avez installé l'archive **PSE.tar**. Si ce n'est pas le cas, consultez le sujet du premier TP.

Placez-vous dans le répertoire **PSE/TP4**.

les processus (73 à 94)

Pratique 1

- exécutez les commandes **ps**, **top**, **pstree** et le **moniteur système** de GNOME
- interprétez leur affichage.

Pratique 2

- parcourez le répertoire **/proc** et consultez quelques fichiers.

Pratique 3

- ouvrez un **terminal** et lancer la commande **emacs &**
- repérez le processus avec la commande **ps xl**
- fermer le **terminal**
- refaites la commande **ps xl**
- que s'est-il passé pour le processus associé à la commande **emacs** ?

Question 1. Dans le programme suivant (sans erreur),

1. indiquez combien il y a de processus ?
2. quels sont les sont les programmes exécutables qu'ils exécutent ?
3. dessinez un arbre généalogique des processus créés

```
#include <...>
int main(void) {
  int p1 = -1, p2 = -1, p3 = -1;
  p1 = fork();
  p2 = fork();
  p3 = fork();
  if (p1 == 0) execv("F1", NULL);
  if (p2 == 0) execv("F2", NULL);
  if (p3 == 0) execv("F3", NULL);
  sleep(5);
  exit(EXIT_SUCCESS);
}
```

Question 2. Dans le programme suivant (sans erreur), si le **PID** du père est **100** et celui du fils **200**, que sera-t-il affiché ?

```
#include <...>
int main(void) {
  int pid, x = 0;
  x++;
  printf("mon pid est %d, x = %d\n", getpid(), x);
  pid = fork();
  if (pid == 0) {
    x++;
    printf("mon pid est %d, x = %d\n", getpid(), x);
  }
  else printf("mon pid est %d, x = %d\n", getpid(), x);
  printf("mon pid est %d, x = %d\n", getpid(), x);
  return EXIT_SUCCESS;
}
```

Pratique 4

- copiez le fichier **exemples/slide086.c** dans le répertoire **TP4** et renommez-le **exercice1.c**
- compilez ce fichier (**make**) et testez-le.
- en particulier, repérez le PID du fils et tapez la commande **ps PID**
- que constatez-vous ?

Exercice 1

- modifiez le programme précédent afin que le processus père attende la fin du processus fils et affiche le code de fin du fils (celui-ci retournera la valeur 111).

Exercice 2

- modifiez le programme de l'exercice précédent, de telle manière qu'il y ait deux programmes distincts : **exercice2_p.c** et **exercice2_f.c**
- vous devrez utiliser **execve()** ou une fonction de sa famille (cf. transparent 88).

Schéma d'exécution classique (transparent 84)

Nous allons mettre en œuvre le « schéma d'exécution classique » vu dans le cours (transparent 84) en réalisant un interprète de commandes (*shell*) simplifié. Le principe général de l'interprète de commandes est le suivant :

```
tantque continuer = VRAI
  code = lire_et_traiter(commande)
  si code != VRAI alors continuer = FAUX
  sinon schéma d'exécution classique (le fils exécute commande)
  finsi
fintantque
```

Vous disposez dans le répertoire **TP4** du module **commande** qui contient la fonction **lire_et_traiter()**, qui lit une commande au clavier et retourne une structure, dont un champ est un tableau de type **argv** directement utilisable dans l'appel système **execv()**. Vous disposez également du squelette **shell.c** que vous devrez modifier dans le prochain exercice.

Exercice 3

- complétez le programme **shell.c** qui réalise l'algorithme ci-dessus.

L'ajout du caractère **&** en fin de ligne de commande permet de changer le schéma d'exécution classique, dans un mode « exécution différée ». Dans ce mode, le *shell* n'attend pas la fin du processus fils : il continue son exécution de manière asynchrone. Le champ **deferred** de la structure retournée par la fonction **lire_et_traiter()**, est positionné à **VRAI** si le caractère **&** a été lu (il doit être placé en fin de ligne) ; il est

retiré du tableau de paramètres.

Lisez les commentaires placés dans le fichier **shell.c**, en particulier la récupération des informations des processus fils qui doit se faire avant l'appel de la fonction **lire_et_traiter()**.

Exercice 4

- modifiez le programme **shell.c** afin de prendre en compte l'exécution différée.