IUT de Montpellier, Dép. Informatique
M2101 - Archi 2 V. Poupet

TD n° 5 - Manipulations de processus

Exercice 1. *Copie du processus courant*

On considère le programme suivant :

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv) {
  int i;
  printf("PID: %d (avant fork)\n", getpid());
  i = fork();
  if (i != 0) {
    printf("père PID: %d, résultat du fork: %d\n",getpid(),i);
  } else {
    printf("fils PID: %d, résultat du fork: %d \n",getpid(),i);
  }
  printf("PID: %d (après fork)\n", getpid());
}
```

1. Recopiez, compilez et exécutez le programme. Expliquez les résultats observés.

2. En utilisant la procédure `unsigned int sleep(unsigned int seconds)` de la bibliothèque `unistd.h`, faites en sorte que les lignes d'affichage du processus fils s'exécutent 5 secondes après celles du processus père.
Que se passe-t-il lorsque l'on exécute le programme ?

Solution :

```
...
  } else {
    sleep(5);
    printf("fils PID: %d, résultat du fork: %d \n",getpid(),i);
  }
...
```

Le père affiche ses résultats immédiatement puis rend le contrôle au *shell*. Quelques secondes plus tard, le fils affiche à son tour ses résultats sur le *shell*.

3. Exécutez le programme de la question précédente puis lancez la commande « `ps -l` » avant que le fils ait fini de s'exécuter (vous avez 5 secondes). Que remarquez-vous ?
Indication : Combien de processus sont en cours, quels sont leurs parents ?

Solution : Le processus père est déjà terminé quand la commande « `ls -l` » est lancée, donc il n'apparaît pas. Il n'y a qu'un seul processus nommé `a.out` (ou le nom de l'exécutable), et son PPID est 1.
En effet, son parent originel (le premier processus lancé) a terminé avant lui, et le fils a donc été récupéré par le processus `init` (dont le PID est 1).

On considère maintenant le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char **argv) {
  int i, j, s, tabpid[3];
  printf("[avant fork] PID: %d\n", getpid());
  for(j = 0; j < 3; j++) {
    tabpid[j] = fork();
    if (tabpid[j] != 0 ) {
      printf("[père] PID: %d, retour fork: %d \n",getpid(),tabpid[j]);
    } else {
      printf("[fils] PID: %d \n",getpid());
      exit(j);
    }
  }
  for(j = 0; j < 3; j++) {
    i= waitpid(tabpid[j], &s, 0);
    if (i > 0) {
      printf("terminé PID: %d\n", i);
    }
    sleep(1);
  }
}
```

4 Que fait ce programme ?

5 Que récupère la variable `s` ? Modifiez le programme pour qu'il affiche le code de retour de chacun des processus fils.
Indication : Il faut chercher comment trouver le code de retour d'un processus dans la documentation de `waitpid`.

Solution : Pour afficher le code de retour d'un processus, on utilise la fonction `WEXITSTATUS` sur l'entier `s` :

```
...
      printf("terminé PID: %d, code: %d\n", i, WEXITSTATUS(s));
...
```

6 Modifiez le programme pour que chacun des fils affiche les valeurs de `tabpid[0]`, `tabpid[1]` et `tabpid[2]`. Observez ce qui se passe lors de l'exécution. Comment l'expliquez-vous ?

Solution :

```
...
      printf("[fils] PID: %d \n",getpid());
      printf("[fils] tabpid: %d, %d, %d", tabpid[0], tabpid[1], tabpid[2]);
      exit(j);
...
```

Le tableau du premier fils ne contient aucune valeur, celui du second contient le PID du premier et celui du troisième contient les PID des deux premiers. Chaque fils reçoit un tableau qui est la copie de celui du père au moment où le `fork` est exécuté, c'est-à-dire qu'il contient bien les résultats des `fork` précédent, mais pas encore celui qui a produit le fils en question.

7 Reprenez le programme `negatif` écrit au TD 4. Rajouttez un appel `fork` juste après avoir recopié l'en-tête du fichier, puis faites en sorte que le processus père génère le négatif de l'image tandis que le fils recopie l'image sans la modifier. Observez le fichier obtenu. Comment expliquez-vous cela ?

8 Exécutez le même programme avec une image en couleur. Expliquez le résultat.

Exercice 2. *Exécution*

1 En utilisant la fonction `execlp`, écrivez un programme qui exécute la commande « `ps -l` ».

Indication : Il faut appeler la commande `execlp` pour remplacer le code du processus courant par celui de la commande `ps`, tout en lui donnant les bons arguments pour lui passer l'option `-l`.

Solution :

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
  execlp("ps", "ps", "-l", NULL);
}
```

2 Écrivez un programme `lancer.c` qui se clone à l'aide de l'appel `fork` et dont le processus fils exécute l'exécutable passé en paramètres avec ses arguments. Par exemple l'appel « `./lancer ls -l *` » doit exécuter la commande « `ls -l *` »

Indication : Il faut utiliser l'appel `fork` puis utiliser une des commandes de la famille `exec` pour remplacer le code du processus fils par celui du programme passé en argument, en lui transmettant également tous les autres arguments.

Solution : Lorsqu'on doit appeler la fonction `exec` avec un nombre variable d'arguments il est plus facile de faire un tableau contenant tous les arguments et de passer ce tableau à la fonction `exec`. On utilise alors les fonctions `execv` ou `execvp` (la version `execvp` est plus pratique car elle cherche l'exécutable dans le *path*).

Le tableau ainsi passé en argument doit se terminer par la valeur `NULL`.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
  int pid, j, status;
  pid = fork();

  if (pid == 0) {
    // processus fils
    // on va créer un tableau de chaînes de caractères (char**)
    // contenant les arguments passés à la fonction main excepté
    // le premier, terminé par NULL
    // (donc args est de longueur argc)
    char** args;
    args = (char**) malloc((argc)*sizeof(char*)); // créer args
    for (j = 0; j < argc-1; j++){
      args[j] = argv[j + 1]; // copier les valeurs de argv dans args
    }
    args[argc-1] = NULL;
    execvp(argv[1], args); // executer
  } else {
    // processus père
    waitpid(pid, &status, 0); // on attend que le fils termine
  }
}
```