

# Programmation Unix

## TD

### Les appels système : gestion de processus

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affichés avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

#### Question 1 Création de processus : la primitive `fork ()`

Soit 2 processus P1 et P2. P1 crée le processus P2 puis exécute une boucle suffisamment longue pour permettre l'observation et dans laquelle il écrit à chaque itération:

```
PERE, mon pid est valeur du pid
Le pid de mon fils est valeur du pid
```

P2 dès sa création, exécute une boucle suffisamment longue pour permettre l'observation et dans laquelle il écrit à chaque itération:

```
FILS, mon pid est valeur du pid
Le pid de mon pere est valeur du pid
```

*Fonctions utiles:*

- `fork()`: cf. cours.
- `getpid()` et `getppid()` retournent respectivement le pid du processus le pid de son père.

**NB** : on n'utilisera ni le `exit` ni le `wait`.

- Écrire le programme C correspondant (respecter l'indentation des messages de P1 et P2 pour davantage de lisibilité sur l'écran au moment de l'exécution)
- Dans une autre fenêtre shell, vérifiez la validité des valeurs des pid affichés par les 2 processus à l'aide de la commande `ps aux | grep votre_nom_de_login`.

## Question 2 Vie des processus

---

- a) Lancez plusieurs fois le programme précédent. Quelles séquences d'exécution observez-vous, que faut-il en déduire ?
- b) Ajustez le nombre d'itérations du processus père P1 de telle sorte qu'il se termine *AVANT* son fils P2. Que constatez-vous ?
- c) Ajustez le nombre d'itérations du processus fils P2 de telle sorte qu'il se termine *AVANT* son père P1. Dans quel état se trouve P2 du point de vue du système ?

## Question 3 Effet du « clonage » sur les données

---

Le processus fils hérite d'une copie des données du père. Cette copie n'est réalisée que lors d'accès en écriture (copy on write) et est telle que :

- l'ensemble des adresses valides des deux processus est le même,
- les valeurs des données dans les deux processus sont les mêmes au retour de l'appel du fork(),
- les données physiques sont différentes

- a) Ecrire le programme suivant de façon à vérifier les affirmations précédentes :

Soit 2 processus P1 et P2. P1 crée le processus P2 PUIS exécute une boucle (suffisamment longue pour permettre l'observation) dans laquelle il décrémente à chaque itération une variable entière X initialisée avec une valeur quelconque:

*PERE valeur de X*

P2 dès sa création, exécute une boucle (nombre d'itérations égal à celui du père) dans laquelle il écrit à chaque itération:

*FILS, valeur de X*

P1 et P2 affiche l'adresse de X

## Question 4 La famille processus

---

Un processus P crée un fils F1 et un fils F2. F2 crée 1 fils F3.

F2 affiche :

"Je suis *w* : F2, le frère de *x*: F1"

F3 affiche :

"Je suis *y* : F3, le petit-fils de *p* : P et le neveu de *x*: F1"

- a) Écrire en C le programme correspondant en sachant que *w* est la valeur du pid de F2, *x* est la valeur du pid de F1, *y* est la valeur du pid de F3 et *p* la valeur du pid de P.

- b) On voudrait aussi que F1 affiche : "Je suis x: F1, le frère de w : F2". Quel problème cela pose-t-il ?  
w est le PID de F2, x est le PID de F1 et y est le PID de F3

### Question 5 La synchronisation père/fils

---

Soit un processus P1 qui crée **d'abord** n ( $n < 5$ ) processus et **ensuite** attend qu'ils terminent.

- a) Chaque  $P_n$  entre dans une boucle (suffisamment longue pour permettre l'observation) puis se termine par `exit(n)`. Chaque fois qu'un de ces fils se termine, P1 affiche :
- le pid du fils terminé,
  - la valeur transmise par le `exit` du fils qui se termine.
  - le nombre de fils restant à attendre
- b) Modifier le programme précédent de façon que le processus père indique également si la terminaison de son fils s'est effectuée :
- normalement et affiche la valeur transmise par le `exit` du fils qui se termine.
  - suite à la réception d'un signal non intercepté et affiche le n° du signal en cause

*Fonctions utiles:*

*fork, wait, exit: cf. cours et le man*

### Question 6 Terminaison des processus

---

Les primitives `exit()` et `_exit()` entraînent la terminaison d'un processus, à la demande de ce dernier.

`exit()` diffère de `_exit()`, entre autres, par le fait qu'elle vide les tampons associés aux fichiers ouverts.

- a) Vérifier, par l'écriture d'un programme, l'affirmation précédente.

`exit()` diffère également de `_exit()` par le fait qu'elle appelle toutes les fonctions dont la demande d'exécution en cas de terminaison a été formulée au moyen de la fonction standard `int atexit(void (*fonction)(void));`.

- b) Écrire un programme qui formule la demande d'exécution de trois fonctions (`func1()`, `func2()` et `func3()`) en cas de terminaison. Le processus doit se terminer sur un appel à `exit()`.
- c) Vérifier, en remplaçant `exit()` par `_exit()` dans le programme précédent, que les fonctions (`func1()`, `func2()` et `func3()`) ne sont pas exécutées à la terminaison du processus

## Question 7 Les temps CPU

---

La primitive `times()` renvoie les temps CPU consommés dans les deux modes (utilisateur et noyau) par le processus ainsi que par ses fils terminés ayant été attendus. Les temps retournés dans les différents champs de la structure `tms` sont exprimés en nombre de clics d'horloge. Pour les convertir en des durées exprimées en secondes, il suffit de les diviser par le nombre de clics d'horloge par seconde obtenu avec `sysconf (_SC_CLK_TCK)`.

Écrire un programme qui :

- 1) affiche le nombre de clics écoulés sur la machine et le nombre clics par seconde
- 2) effectue un certain calcul assez long (> 1 seconde)
- 3) affiche les temps CPU utilisateur et système consommés

## Question 8 Les groupes de processus

---

La primitive `pid_t getpgrp(void)` renvoie le PID du leader du groupe de l'appelant.

- a) Ecrire un programme qui affiche le PID du processus leader du groupe auquel il appartient.

Un processus appartient au même groupe que celui auquel appartient son père et ceci jusqu'à ce qu'il demande à en changer.

- b) Modifier le programme précédent de façon qu'il crée un processus fils qui affiche aussi le PID du processus leader du groupe auquel il appartient.
- c) Vérifier si les deux processus précédents appartiennent au même groupe ?
- d) Appartiennent-ils au même groupe auquel appartient le shell à partir duquel vous avez demandé l'exécution (`ps j`) ?

Lorsqu'un processus leader d'un groupe de processus se termine, un signal `SIGHUP` est émis à chaque processus stoppé membre du groupe.

- e) Modifier le programme précédent de façon que le processus fils ne se termine pas immédiatement (utiliser `pause()` à la fin du code du fils) et mettez le à l'état "stoppé".
- f) Que devient le processus fils une fois que le processus père (leader de son groupe) s'est terminé ?

La primitive `pid_t setpgid(pid_t pid, pid_t pgid)` rattache le processus `pid` au groupe `pgid`.

- g) Modifier le programme précédent de façon que le processus fils change de groupe par rapport à son père.
- h) Que devient le processus fils une fois que le processus père (leader de son groupe) s'est terminé ?