TP1 : fork, exec et signaux

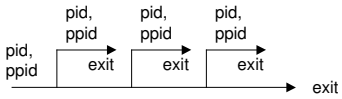
RS : Réseaux et Systèmes

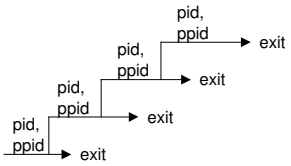
Rappel : Il est indispensable d'ajouter les drapeaux `-Wall -Werror -Wformat` pour profiter des vérifications de base à la compilation. En effet, `gcc` accepte par défaut du code manifestement faux sans le moindre avertissement, ce qui mène à des erreurs difficiles à trouver.

★ Exercice 1: Appel système `fork`.

▷ **Question 1:** Écrire un programme qui crée 10 processus fils. Chacun d'entre eux devra afficher dix fois d'affilé son numéro d'ordre entre 0 et 9. Vérifiez que votre programme affiche 100 caractères.

▷ **Question 2:** Reprise de la Question 4 du TD1
On considère les deux structures de filiation (chaîne et arbre) représentées ci-après. Écrire un programme qui réalise une chaîne de `n` processus, où `n` est passé en paramètre de l'exécution de la commande (par exemple, n = 3 sur la figure ci-dessus). Faire imprimer le numéro de chaque processus et celui de son père. Même question avec la structure en arbre.





Dans le cas de la filiation en arbre, il est tout à fait normal que l'invite du shell apparaisse au milieu des affichages des fils. C'est parce que le shell l'affiche dès que le processus qu'il a lancé lui-même termine. Et dans cette filiation, le père termine avant le fils. On peut continuer à utiliser le shell comme si de rien n'était, ou tout effacer d'un Ctrl-L.
De même, il est normal que le PID indiqué pour le père soit parfois 1 dans la filiation en arbre. Cela se produit quand le père a quitté avant que son fils n'affiche son ppid. Dans ce cas, le fils est déjà rataché au processus init, de pid 1.

★ Exercice 2: Appel système exec.

▷ **Question 1:** Reprise de la Question 8 du TD1 Écrire un programme `doit` qui exécute une commande Unix que l'on lui passe en paramètre. *Exemple :* `doit ls -lt /`

▷ **Question 2:** Écrire un programme `multido` qui exécute au plus cinq fois (dans des processus séparés) une commande Unix que l'on lui passe en paramètre. Si l'une des exécutions provoque une erreur, il ne faut pas réaliser les exécutions suivantes.
Pour tester votre travail, vous pouvez utiliser la commande `./multido mkdir toto` car `mkdir toto` renvoit un code d'erreur si le répertoire à créer existe déjà.

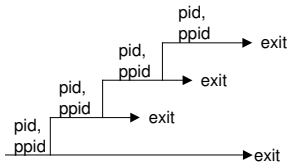
★ Exercice 3: Signaux.

Rappel : nous utilisons l'interface POSIX. Évitez donc les tutos Internet qui utilisent la fonction `signal()`.

▷ **Question 1:** Écrire un programme ne se terminant qu'au cinquième Ctrl-C.

▷ **Question 2:** Écrire un programme créant deux fils, en envoyant le signal SIGUSR1 à son fils cadet. À la réception de ce signal, le fils cadet devra envoyer le signal SIGUSR2 au fils ainé (qui provoque sa terminaison) avant de s'arrêter.
Indication : il est très difficile d'interchanger les rôles des deux fils.

▷ **Question 3:** Seconde reprise de la Question 4 du TD1.
Modifiez votre filiation en arbre afin que le premier père attende le dernier fils, mais que les pères intermédiaires quittent au plus tôt.
Indication : cette question est placée dans un exercice nommé «Signaux».



★ Exercice 4: Réimplementer if.

▷ **Question 1:** Écrire une version simplifiée du programme `if` que vous nommerez `si`.
Testez votre travail avec la commande suivante : `./si test -e toto alors echo present sinon echo absent`