Les processus sous Unix (programmation)

Vincent Danjean

IUP L3 — Sytèmes

Résumé
Création et manipulation de processus avec le langage C.

1 Manipulations des processus en langage C

Note: compilation
Pour compiler vos programmes C, la ligne de compilation à utiliser est la suivante :
`gcc -g -Wall -Werror` *`programme.c`* `-o` *`programme`*

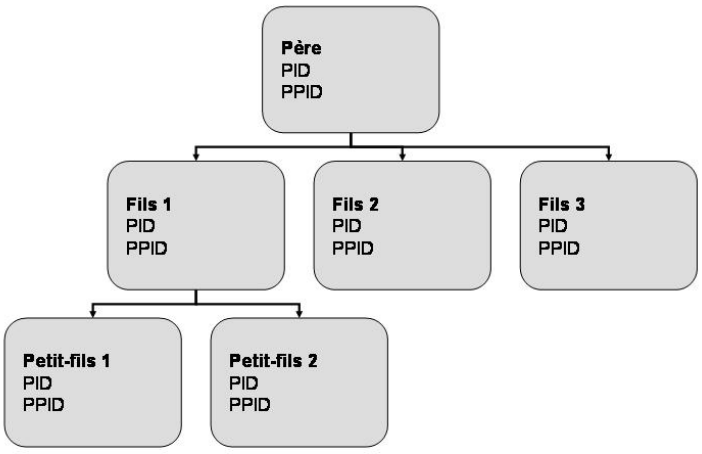
Note: travail de documentation
Une partie non négligeable du travail de ce TP consiste à lire et comprendre les parties importantes des pages de manuels des fonctions citées. Votre enseignant est là pour vous aider à déchiffrer cette documentation si nécessaire. N'hésitez pas à lui poser des questions !

1.1 Identification des processus

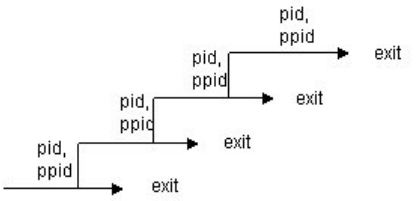
1. Écrivez un programme qui affiche le numéro (PID) du processus qui l'exécute. Lancez ce programme plusieurs fois ; que remarquez vous ?
2. Modifiez ce programme pour qu'il affiche son PID et son PPID. Exécutez le plusieurs fois ; que remarquez vous ?

1.2 Création de processus

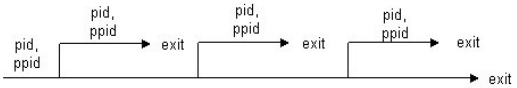
1. Écrire un programme qui reproduit l'arbre généalogique ci-après. Chaque processus doit afficher son PID, son PPID, et afficher les PIDs des fils qu'il engendre.



2. Que se passe-t-il si on lance plusieurs fois le programme ? (observez l'ordre d'apparition des messages à l'écran et commentez)
3. On considère deux structures de filiation (chaîne et arbre) représentées ci-dessous. Écrire un programme qui réalise une chaîne de n processus, où n est passé en paramètre de l'exécution de la commande (par exemple, $n = 4$ dans l'exemple donné). Faire imprimer le numéro de chaque processus et celui de son père. Même question avec la structure en arbre.



Structure en arbre



Structure en chaîne

1.3 Synchronisation élémentaire de processus

1. Modifiez le programme de la question 1.2.1 pour que le fils 2 affiche son message avant les fils 1 et 3. Est-ce que la solution peut être garantie ?
2. Modifiez le programme de la question 1.2.1 pour que les petits-fils 1 et 2 affichent leur message avant les fils 2 et 3. Est-ce que la solution peut être garantie ?

1.4 Exécution de programmes

La primitive `execv` permet de changer le programme (i.e. la suite d'instructions) exécuté au sein d'un processus. Le nouveau programme est chargé à partir d'un fichier (binaire exécutable) et son exécution commence alors au début de sa fonction `main` (avec les paramètres éventuels passés à la fonction `execv`).

Note: *Il **n**'y a **pas** de création de nouveau processus avec* ***execv****.*

prototype de la fonction `execv`

```
#include <unistd.h>
int execv(char* filename, char* argv[]);
```

Le paramètre `filename` pointe vers le nom (absolu ou relatif) du fichier exécutable, `argv` vers le tableau contenant les arguments (terminé par `NULL`). Par convention, le paramètre `argv[0]` contient le nom du fichier exécutable, les arguments suivants étant les paramètres successifs de la commande. (Voir aussi les primitives `execve` et `execvp`).

1. Que fait le programme ci-dessous.

```
#include <stdio.h>
#include <unistd.h>
#define NMAX 5
int main()
    char* argv[NMAX];
    argv[0] = "ls"; argv[1] = "-lt"; argv[2] = "/"; argv[3] = NULL;
    execv("/bin/ls", argv);
```

2. Écrire un programme `execcmd` qui exécute une commande Unix passée en paramètre. Exemple d'exécution : `execcmd /bin/ls -Ft /`

2 Pour aller plus loin : un shell à soi

Un shell est un programme qui attend des commandes sur son entrée standard et qui les exécutes. C'est ce que vous avez dans vos terminaux.
Les shells installés sur les systèmes à votre disposition sont très complexes. Ils peuvent gérer des redirections complexes, manipuler des jobs (avant-plan/arrière-plan), gérer des variables, permettre des manipulations de texte (double quote, backquote, simple quote, ...), etc.
Le but de cet exercice (qui sera poursuivi dans une prochaine séance) est de créer un shell minimaliste vous permettant de lancer des commandes simples.

2.1 Programme principal

Compiler [1], tester et comprendre le programme `shell.c`. Il permet de lancer une commande introduite par l'utilisateur : vous pouvez par exemple taper "ls" ou encore "ls -l".

Note: Utilitaires
Des fonctions pour vous aider sont disponibles dans le fichier `shell-utils.c` (avec les entêtes dans le fichier `shell-utils.h`). Vous pouvez utiliser ces fonctions, mais il ne faut pas modifier ces fichiers a priori.

2.2 Extensions

1. Faite afficher un prompt (ie une invite de commande, par exemple "commande : ") pour que l'utilisateur voit quand il peut taper une commande.
2. Modifier ce programme pour qu'il lance des commandes tant que l'utilisateur n'aura pas tapé la commande `exit`.
3. Faire en sorte que ce shell attendre la fin de la commande précédente avant de demander la commande suivante.
4. La commande `cd` n'est pas un programme mais une commande interne des shells. Pourquoi ? Implémentez une telle commande dans votre shell (voir l'appel système `chdir`).

1. Pour compiler le programme `shell` fourni, utiliser la commande "`make shell`" : elle prendra automatiquement en compte les fonctions du fichier `shell-utils.c`.