

Les processus sous Unix

Vincent DANJEAN

Pierre Louis AUBLIN

L3 MIAGE — Système Réseaux

Résumé

Observation et manipulation de processus avec le shell et le langage de programmation C.

Une partie de ce sujet est inspiré de la page web : <http://www.tuteurs.ens.fr/unix/processus.html>

1 Manipulations des processus avec le shell

Unix est un système multi-tâches, c'est-à-dire qu'il peut exécuter plusieurs programmes à la fois. Un processus est une instance d'un programme en train de s'exécuter, une tâche. Le shell crée un nouveau processus pour exécuter chaque commande.

1.1 Mettre un processus en tâche de fond

Si on lance une commande qui prend beaucoup de temps (comme un calcul, ou une nouvelle fenêtre), on peut l'interrompre par **Control-C**. Ceci interrompt (définitivement) la commande. On peut aussi exécuter une commande en tâche de fond. Le shell rend alors la main avant la fin de la commande. Pour le faire, on ajoute un **&** à la fin de la commande.

Exemple:

```
danjeanv@mandelbrot:~> gedit fichier.c &
danjeanv@mandelbrot:~> xpdf sujet.pdf &
```

Dans le premier exemple, on lance un éditeur graphique en parallèle avec le shell ; dans le second exemple, on met le lecteur de fichier **pdf** en tâche de fond, ce qui permet de le garder ouvert tout en continuant de taper d'autres commandes.

On reprend la main immédiatement, sans attendre la fin de l'exécution de la commande. On peut donc taper d'autres commandes dans le même terminal, pendant que la précédente s'exécute.

1.2 background et foreground

Comme on vient de le voir, si vous avez pensé à terminer votre ligne de commande par une esperluette, le programme que vous avez lancé tourne en arrière-plan (background, abrégé en **bg**). Si vous avez omis l'esperluette, le programme prend la précedence sur le shell. On dit qu'il tourne au premier plan (foreground, abrégé en **fg**). Les lignes tapées au clavier sont mémorisées mais ne seront pas exécutées par le shell avant que le programme en cours d'exécution n'ait fini son calcul. Vous pouvez malgré tout faire passer ce programme en tâche de fond, grâce à la manipulation suivante :

Exemple:

```
danjeanv@mandelbrot:~> xpdf sujet.pdf
^Z
suspended
danjeanv@mandelbrot:~> bg
[1] xpdf sujet.pdf &
danjeanv@mandelbrot:~>
```

`^Z` est un signal intercepté par le shell, qui suspend l'exécution du programme sans détruire le processus correspondant. Les calculs déjà effectués par ce programme ne sont pas perdus. Dans l'exemple précédent, si on demande à un `xpdf` suspendu de changer de page (SPC), il ne le fera pas, mais il se souviendra de le faire dès qu'il aura à nouveau accès aux ressources de l'ordinateur.

À partir de l'état suspendu, on peut faire passer un programme :

- au premier plan, en tapant `fg`;
- en arrière-plan, en tapant `bg`.

Son exécution reprend alors là où on l'avait laissée.

Quand il n'y a qu'un seul programme en arrière-plan dans le terminal courant, on peut le faire passer au premier plan en tapant `fg`. Cela permet en particulier d'interrompre son exécution grâce à `^C`, que la plupart des programmes comprennent. `^C` n'affecte que l'éventuel unique programme qui tourne au premier plan dans le terminal où il est tapé. Quand il y en a plusieurs, c'est un peu plus compliqué, mais c'est bien sûr possible.

Vous pouvez pratiquer `^C`, `^Z`, `fg`, `bg` de façon visuelle et didactique en lançant la commande

Exemple:

```
xclock -update 1
```

et en observant attentivement les secondes (pensez à en laisser quelques-unes s'écouler). Vous remarquerez que ce programme ne se met plus à l'heure lorsqu'il est suspendu.

Résumé :

Le programme tourne au premier plan :

`^Z` le suspend ;
`^C` l'interrompt.

Le programme est suspendu :

`fg` le passe au premier plan ;
`bg` le passe en arrière-plan.

Le programme tourne en arrière-plan :

`fg` le passe au premier plan ;

Le programme ne tourne pas :

il n'y a rien à faire...

Note: `fg` et `bg` acceptent un numéro de job en argument pour distinguer les différents processus s'il y en a plusieurs suspendus ou en arrière-plan. C'est le numéro affiché par le shell après un lancement en arrière-plan (avec `&`) ou une suspension (avec `^Z`) :

```
mandelbrot ~ $ fg %2
```

1.3 Voir les processus

La commande `ps` montre où en sont les tâches de fond :

Exemple:

```
danjeanv@mandelbrot:~> ps t
  PID TTY          STAT       TIME CMD
 4450 pts/0    Ss           00:00:00 /usr/local/bin/lcsh
 4782 pts/0    Ss           00:00:02 xpdf sujet.pdf
 4841 pts/0    R+          00:00:00 ps t
```

`ps` affiche la liste des processus que vous avez lancés :

PID (process identifier) : c'est le numéro du processus.

TTY : indique le terminal dans lequel a été lancé le processus. Un point d'interrogation signifie que le processus n'est attaché à aucun terminal (par exemple les démons).

STAT : indique l'état du processus (option 't') :

R : actif (running)

S : non activé depuis moins de 20 secondes (sleeping)

I : non activé depuis plus de 20 secondes (idle)

T : arrêté (suspendu)

Z : zombie

TIME : indique le temps machine utilisé par le programme (et non pas le temps depuis lequel le processus a été lancé!).

La commande `ps` a différentes options, dont les suivantes :

a (all) : donne la liste de tous les processus, y compris ceux dont vous n'êtes pas propriétaire.

g (global, général...) : donne la liste de tous les processus dont vous êtes propriétaire.

u (user, utilisateur) : donne davantage d'informations (nom du propriétaire, heure de lancement, pourcentage de mémoire occupée par le processus, etc.).

x : affiche aussi les processus qui ne sont pas associés à un terminal.

w : ne tronque pas à 80 caractères (peut être utilisée plusieurs fois pour tronquer plus loin)

`ps agux` est en fait souvent utilisé pour avoir des informations sur tout les processus.

Note: BSD et System V Il existe deux grandes familles d'Unix : BSD et System V. Les deux systèmes ont à peu près les mêmes commandes et fonctions, mais il existe quelques variations. En particulier, pour la commande `ps`, les options ne sont pas les mêmes sur les deux systèmes. Ainsi, sur Solaris (System V), pour avoir des informations sur tous les processus, on utilisera plutôt : `ps -elf`. Se référer à la page `man` de la commande pour plus d'information sur les options disponibles dans ce cas.

La commande `top` affiche les mêmes informations, mais de façon dynamique : elle indique en fait par ordre décroissant le temps machine des processus, les plus gourmands en premier. Ça permet par exemple de savoir pourquoi une machine rame, et c'est très instructif sur les processus gourmands en ressources...

1.4 Tuer les procesus

Les programmes ont tous une commande spécifique pour les quitter (q, menu où cliquer, etc). C'est seulement dans le cas où vous ne parvenez pas à les quitter correctement, pour une raison ou une autre, que vous pouvez avoir besoin de tuer le processus.

Avant toute chose, essayez de taper `^C` ou `^D`. Si cependant ça ne marche pas, utilisez la commande `kill [-SIGNAL] pid...`

Exemple:

```
danjeanv@mandelbrot:~> ps
  PID TTY          TIME CMD
 4450 pts/0    00:00:00 /usr/local/bin/tcsh
 4782 pts/0    00:00:02 xpdf sujet.pdf
 4841 pts/0    00:00:00 ps
danjeanv@mandelbrot:~> kill 4782
danjeanv@mandelbrot:~> ps
  PID TTY          TIME CMD
 4450 pts/0    00:00:00 /usr/local/bin/tcsh
 4846 pts/0    00:00:00 ps
```

`kill -9 pid` tue le processus à tous les coups. Vous ne pouvez tuer que les processus dont vous êtes propriétaire. Il convient de ne l'utiliser que si aucune autre méthode ne marche, car dans ce cas le programme n'a aucun moyen de faire quoi que ce soit avant de mourir.

Note: Si on ne précise pas de numéro de signal, c'est le signal 15 (**TERM**) qui est envoyé. Essayer d'envoyer les signaux **STOP** et **CONT** pour suspendre et reprendre un processus. Est-ce que ça vous permet de reprendre la main dans le shell ? Expliquez.

1.5 Environnement des processus

1. Testez la commande `env` pour afficher les valeurs des variables d'environnement courantes.
2. Avec la commande `echo`, examinez en particulier la valeur de la variable `PATH`. Quelle est sa valeur¹ ? Cette variable permet de localiser automatiquement un fichier exécutable (binaire) en définissant une suite de répertoires dans lesquels on recherche le programme exécutable, dans l'ordre d'apparition des répertoires. C'est ce qui permet, par exemple, à l'utilisateur de taper la commande `gcc` au lieu (par exemple) de `/usr/local/bin/gcc`. Pour connaître la localisation exacte d'une commande, on utilise `which`. Essayez, par exemple, `which ls` et `which gcc`. Quels sont les résultats obtenus ?

Les variables d'environnement peuvent être modifiées par le shell. Par défaut, un processus hérite à sa création des variables de son père. Une fois un processus créé, seul ce processus peut modifier ses propres variables. Cela signifie que si on modifie une variable d'environnement dans un shell, cette modification sera visible pour les processus créés dans ce shell ultérieurement (mais pas par les processus créés par ce shell antérieurement ni par les processus créés à partir d'autres shells). Il existe deux familles principales de shells : ceux basés sur `bash` et ceux basés sur `tcsh`. Suivant la machine (ou même la configuration de la machine) sur laquelle vous vous trouvez², utilisez les bonnes commandes.

Shell `tcsh`

1. Exécutez la commande `setenv SAVEPATH $PATH` (qui recopie dans la nouvelle variable d'environnement `SAVEPATH` la valeur courante de `PATH`). Puis exécutez `setenv PATH /:.` et tapez la commande `ls`. Que constatez-vous ? Comment l'expliquez-vous ? Pour restaurer la valeur de votre variable `PATH`, exécutez `setenv PATH $SAVEPATH`. Puis exécutez `unsetenv SAVEPATH` pour éliminer la variable `SAVEPATH` de votre environnement.
2. Comment se fait-il que la commande `setenv` continue de fonctionner malgré la modification du `PATH` ?

Shell `bash`

1. Il faut taper `echo $PATH` pour voir le contenu de la variable `PATH`.
2. Vous pouvez taper `echo $SHELL` pour voir quel shell vous utilisez.

1. Exécutez la commande `export SAVEPATH=$PATH` (qui recopie dans la nouvelle variable d'environnement `SAVEPATH` la valeur courante de `PATH`). Puis exécutez `export PATH=/. :` et tapez la commande `ls`. Que constatez-vous ? Comment l'expliquez-vous ? Pour restaurer la valeur de votre variable `PATH`, exécutez `export PATH=$SAVEPATH`. Puis exécutez `unset SAVEPATH` pour éliminer la variable `SAVEPATH` de votre environnement.
2. Comment se fait-il que la commande `export` continue de fonctionner malgré la modification du `PATH` ?

2 Manipulations des processus en langage C

Note: compilation

Un `Makefile` vous est fourni pour vous aider à compiler vos programmes. Il suffit de taper `make programme` pour compiler un programme et `make clean` pour effacer les fichiers compilés. Si vous rajoutez des programmes, vous pouvez le modifier en prenant exemple sur ce qui existe déjà.

Note: travail de documentation

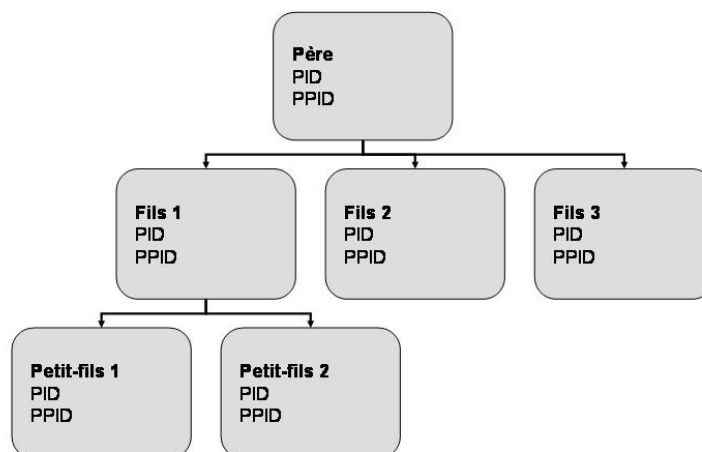
Une partie non négligeable du travail de ce TP consiste à lire et comprendre les parties importantes des pages de manuels des fonctions citées. Vos enseignants sont là pour vous aider à déchiffrer cette documentation si nécessaire. N'hésitez pas à leur poser des questions !

2.1 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 ?

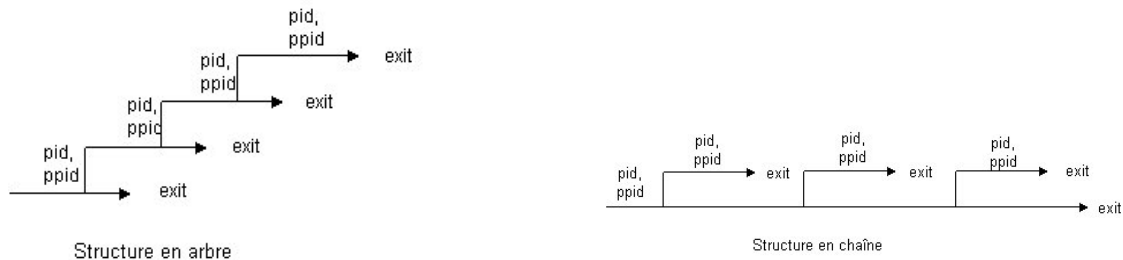
2.2 Création de processus

1. Écrire un programme qui reproduit l'arbre généalogique ci-après. Chaque processus doit afficher son PID, son PPID et afficher les 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)

- 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.



2.3 Synchronisation élémentaire de processus

- Modifiez le programme de la question 2.2.1 pour que le fils 2 affiche son message avant les fils 1 et 3.
- Modifiez ce programme pour que les petits-fils 1 et 2 affichent leur message avant les fils 2 et 3.

2.4 Exécution de programmes

La primitive `execv` permet de créer un processus pour exécuter un programme déterminé (qui a auparavant été placé dans un fichier, sous forme de binaire exécutable).

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 la primitive `execve`).

- 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);
}
```

- Écrire un programme `execcmd` qui exécute une commande Unix passée en paramètre. Exemple d'exécution : `execcmd /bin/ls -Ft /`
- Écrire un programme qui lance le programme de la question 2.1.2, en modifiant certaines variables d'environnement. Faire afficher ces variables par le programme. (utiliser la primitive `execve`).