

IF 110 : Système d'exploitation - TP3

Claire Capdevielle - Rémi Giraud
ccapdevi@labri.fr, remi.giraud@labri.fr

2015-2016

1 La notion de processus Unix

Dans cette partie vous vous familiariserez avec la notion de **processus** sous Unix. Celle-ci est importante, voir essentielle, pour comprendre le mécanisme fondamental de fonctionnement de ce système d'exploitation. Notamment vous verrez que sous Unix toute commande est rattachée à un processus. Vous vous familiariserez aussi avec les **mécanismes de redirection** et vous verrez que ces mécanismes sont étroitement liés aux processus.

Généralités. Tout d'abord, vous ne devez pas oublier que le système Unix est **multi-utilisateurs**¹ et **multi-tâches** (i.e. vous pouvez lancer sous une même session de travail plusieurs tâches qui s'exécuteront simultanément²). Cela signifie que, sous le même système, et à un instant donné plusieurs utilisateurs peuvent lancer des programmes exécutables différents ou même identiques. Dans ce cas, il faut bien que le système puisse faire la différence entre tous ces programmes. C'est pour cela qu'a été introduite la notion de processus.

Q1.1 Sur quelle machine s'exécutent les commandes que vous invoquez dans votre terminal? (Vous utiliserez `hostname`)

Définition d'un processus. Désormais plutôt que de parler de tâche qui est un terme un peu flou, nous parlerons de processus.

Définition : Un programme est un fichier exécutable, et un processus est une occurrence d'un programme en exécution.

Les processus s'exécutent indépendamment les uns des autres. Pour parvenir à cette caractéristique, le noyau³ du système attribue un identificateur à chaque processus. Cet identificateur de processus, un entier, est appelé pid pour "process identifier". Les processus ont en quelque sorte une vie ; ils naissent (lorsque vous lancez un programme ou une commande), ils vivent (lorsqu'ils s'exécutent) et meurent (lorsqu'ils ont terminé leur exécution). À leur naissance, chaque processus se voit attribuer par le système un pid distinct. Ainsi, à un instant donné, tous les processus vivants ont un pid différent, et il n'y a pas de conflit pour le système qui est alors capable de différencier tous les processus via leur identificateur⁴. Enfin, lorsqu'un processus meurt, son pid redevient disponible et pourra être réattribué par le système à un nouveau processus.

1. C'est pour cela que vous devez vous connecter via un login et un password.

2. Sur les machines qui ne possèdent qu'un seul processeur de calcul, le terme simultanément est un abus de langage. Sur ces machines les différentes tâches s'effectuent en réalité en temps partagé.

3. "Kernel" en anglais

4. Même si deux processus sont deux occurrences d'un même programme, ils ont des pid différents.

2 Première vision des processus

Q1.2 Lancez la commande `xload`. Avez-vous toujours le contrôle dans le terminal depuis lequel vous avez lancé `xload` ?

Q1.3 Testez la commande `Ctrl-z` dans votre terminal. Que se passe-t-il ?

Q1.4 Testez la commande `jobs`. Qu'indique-t-elle ?

Q1.5 Testez la commande `fg`. Interrompez à nouveau le processus.

Q1.6 Testez la commande `bg`. Que se passe-t-il ? Avez-vous toujours le contrôle dans le terminal ?

Q1.7 Lancez à nouveau la commande `xload`. Interrompez-la.

Q1.8 Qu'indique maintenant la commande `jobs` ?

Q1.9 Que provoque la commande `kill -9 %1` ?

Q1.10 Testez la commande `ps`. Avez-vous trouvé les pid de vos processus ?

Q1.11 Testez la commande `ps -f`. Quelles sont les différences par rapport à la commande précédente ?

Q1.12 Quel est le numéro du processus `xload` restant ?

Q1.13 Testez la commande `ps -af`.

Q1.14 Essayez de tuer (par la commande `kill -9 pid`⁵) un processus d'un autre utilisateur. Que se passe-t-il ?

Q1.15 Tuez votre processus `xload` restant en utilisant cette fois-ci son numéro de processus (pid).

Q1.16 Lancez la commande `xload &`. Avez-vous toujours le contrôle dans le terminal ?

Q1.17 Faites passer le processus en avant-plan par la commande `fg`.

Q1.18 Faites à nouveau passer le processus en arrière-plan avec la commande `bg`.

3 Mécanismes de redirection

Chaque processus a une entrée standard, une sortie standard et une sortie d'erreur. Par défaut, l'entrée standard est le clavier, la sortie standard est l'écran, et la sortie d'erreur est aussi l'écran. Il est possible de modifier ce comportement à l'aide d'opérateur de redirection :

- Rediriger la sortie standard (resp. sortie d'erreur) d'un processus vers un fichier : `>` (resp. `2 >`)
- Ajouter la sortie standard (resp. sortie d'erreur) d'un processus à un fichier, sans effacer ce qui précède : `>>` (resp. `2 >>`)
- Rediriger l'entrée standard d'un processus. Le processus lira alors le fichier au lieu du clavier : `<`
- Connecter directement la sortie standard d'un processus sur l'entrée standard d'un autre processus : `|`

Q1.19 Lancez la commande `ls` depuis votre répertoire personnel.

Q1.20 Relancez la commande en redirigeant cette fois-ci la sortie standard dans un fichier dont le nom est `resultats`. Pour cela, utilisez la commande `ls > resultats`.

Q1.21 Éditez le fichier `resultats`. Que contient-il ?

Q1.22 Donnez une commande qui permet d'écrire dans un fichier (à l'aide de `echo`).

Q1.23 En une commande, faites afficher le nombre de dossiers et de fichiers que contient `/usr/bin`. (Vous utiliserez les commandes `ls`, `wc` ainsi qu'un pipe (i.e. opérateur : `|`)).

5. La commande `kill` sert à envoyer un signal à un processus. Elle est souvent utilisée pour tuer un processus en lui envoyant le signal `KILL` (man `kill`)