Architecture d'un OS : Introduction à Linux (suite)

TP INFO-F201 : séance 4

Keno Merckx, Cédric Ternon et Jacopo De Stefani

ULB

2015-2016

1 Le *shell* (suite)

1.1 Redirection d'I/O

A chaque processus, sont associés 3 I/O standard :

1. `stdin` : le *standard input*
2. `stdout` : le *standard output*
3. `stderr` : le *standard error*

Le *shell* vous permet de rediriger ces I/O vers des fichiers ou d'autres processus :

- `>` : vous permet de rediriger le `stdout` d'un processus vers un fichier. Le contenu du fichier est écrasé (perdu !). Par exemple, `ls *.c > foo` listera les fichiers se terminant par `.c` du répertoire courant. Le résultat sera stocké dans le fichier `foo`.
- `>>` : idem que `>` mais le fichier n'est pas écrasé. S'il existe, le résultat de la commande est ajouté à la fin du fichier.
- `2 >` : idem que `>` mais pour le `stderr`.
- `2 >>` : idem que `>>` mais pour le `stderr`.
- `<` : vous permet de rediriger le `stdin` vers un fichier. Par exemple, si vous lancez un programme "fait maison" de cette marnière : `./a.out < bar`, lorsque votre programme exécutera un `scanf`, au lieu d'attendre le résultat au clavier, il lira l'information dans le fichier `bar`.
- `|` (appelé *pipe*) : vous permet de rediriger le `stdout` vers le `stdin` d'un autre processus. Souvenez-vous des commandes `less` et `grep`. Si vous ne passez pas de fichier en paramètre, ces commandes attendent leur input sur le `stdin`. Ainsi, `ps -ef | less` vous permettra de lister tous les processus et de vous balader dans cette liste en utilisant les fonctionalités de `less`.

1.2 Exercices

1. Rédigez un programme C qui affiche votre nom à l'écran, et compilez-le en un exécutable appelé `name`.
2. Essayez de le lancer en utilisant la commande `name`. Cela fonctionne-t-il ? Si oui, pourquoi ? Si non, que faut-il modifier pour que cela fonctionne ?
3. De quelle autre façon pouvez vous lancer votre programme ?
4. Utilisez votre programme (sans modification) pour écrire votre nom dans un fichier ?
5. Rédiger un programme C qui demande votre nom au clavier et qui vous dit bonjour et compilez-le en un exécutable appelé `hello`.
6. Comment pouvez-vous combiner vos deux programmes en une commande qui vous dit bonjour directement ?

2 L'arborescence des fichier (suite)

2.1 Le système de permissions

Sous UNIX, à chaque fichier est associés un propriétaire (l'un des utlisateurs qui a un compte sur la machine) ainsi qu'un groupe (un groupe d'utilisateurs de la machine). Ces informations peuvent être consultées grâce à la commande `ls -l`.

De plus, pour chaque fichier, on distingue trois types de *permissions*, la permission en *lecture* (read), en *écriture* (write), et en *execution* (execute). Pour un fichiers de données, les permissions intéressantes sont la lecture et l'écriture, qui permettent de consutler et de modifier le fichier. Pour les fichiers exécutables, la permission en exécution permet de lancer l'exécution du programme. Quant aux répertoires, la permission en lecture permet de consulter le répertoire (par la commande `ls` notamment), la permission en écriture permet de créer des fichiers dans ce répertoire, et la permission en exécution permet de le *traverser*, ç.-à-d. soit de se positionner dedans grâce à `cd`, soit de le traverser (au travers d'un chemin de type "`repertoire1/repertoire2/fichier`") pour accéder à l'un des fichiers qu'il contient.

Les systèmes UNIX maintiennent, pour chaque fichier, 3 ensemble de permissions : les permissions accordées au propriétaire du fichier (*user*), celles accordées au groupe du propriétaire du fichier (*group*), et celles accordées à tous les autres utilisateurs (*other*). Pour chacune de ces entités, on peut avoir comme permissions n'importe quel sous-ensemble de l'ensemble {`r,w,x`} (`r` pour lecture, `w` pour écriture, `x` pour exécution). On peut consulter ces permission à l'aide de la command `ls -l`.

2.2 Consultation et modification des permission

2.2.1 `ls -l`

Cette commande permet de lister le contenu d'une directory avec les permissions, ainsi que le propriétaire et le groupe de chaque fichier. Pour chaque fichier, les permissions sont affichées sous le format suivant : permissions du propriétaire, permissions du groupe, permissions des autres. Pour chacune de ces trois entités, les permissions se composent de trois caractères dénotant, dans l'ordre, si l'utilisateur a les permissions en lecture, écriture, et exécution. Le caractère "`r`", "`w`", ou "`x`" dénote la présence d'une permission, et le caractère "`-`" son absence.

2.2.2 `chown`

La commande `chown` (*change owner*) permet de changer le propriétaire d'un fichier. Seul le propriétaire actuel du fichier (ou le super-user) peut l'exécuter. Sa syntaxe est `chown nouveau_proprietaire fihcier1 fichier2 ...`.

2.2.3 `chgrp` et `groups`

La commande `chgrp` (*change group*) permet de changer le groupe associé à un fichier. Sa syntaxe est `chgrp nouveau_groupe fichier1 fichier2 ...`, où `nouveau_groupe` est soit le numéro du nouveau groupe soit son nom. La commande `groups user` permet de consulter le groupe auquel appartient l'utilisateur `user`.

2.2.4 `chmod`

Le command `chmod` (*change mode*) permet de changer les permissions d'un fichier. Elle n'est accessible qu'au propriétaire du fichier ou au super-user. Sa syntaxe est `chmod mode fichier1 fichier2...`, où `mode` est de la forme {`a`,`u`,`g`,`o`}{`+`,`-`}{`r`,`x`,`w`}

Les lettres `a`, `u`, `g`, `o` dénotent respectivement tous les utilisateurs (*all*, le propriétaire (*user*), le groupe du propriétaire (*group*), et les autres (*others*), c'est-à-dire ceux qui ne sont ni le propriéatire et qui n'appartiennent pas a son groupe. On peut en spécifier plusieurs. Le caractère "`+`" dénote qu'on veut rajouter des permissions, le caractère "`-`" qu'on veut en supprimmer.

Finalement, on spécifie à quelles permissions précisément la commande s'applique, en spécifiant une ou plusieurs lettres parmi {`r`,`x`,`w`}.

Par exemple, la commande `chmod ug+rw brol` permet d'ajouter au fichier "`brol`" les permissions en lecture et en écriture, pour le propriétaire et le groupe du propriétaire.

2.3 Exercices

1. Se déplacer dans le répertoire `/home` et lire son contenu. Y créer un autre répertoire. Cela marche-t-il ? Pourquoi ? Si non, pouquoi avez-vous quand-même pu vous positionner dans ce répertoire et en lire son contenu ?
2. Essayez de vous placer dans la home directory de l'un de vos camarades. Cela fonctionne-t-il ? Pourquoi ? Essayez de lister le contenu de cette directory ? Le pouvez-vous ? Pourquoi ?
3. Supposons que vous vouliez avoir un sous-répertoire public dans votre home directory, dans laquelle vos camarades pourraient écrire des fichiers à votre attention, comment procéderiez-vous ? Imaginez et testez une solution. Vos amis peuvent-ils découvrir par eux-mêmes le nom de ce répertoire ou doivent-ils le connaître à l'avance ? Pourquoi ? Comment modifier cela ? Testez.

3 Les processus

3.1 Introduction

Un processus est un programme en cours d'exécution. Sous Unix en général et sous Linux en particulier, à chaque processus est associé un nombre entier l'identifiant. Ce nombre est appelé le PID (*process ID*) du processus. Le premier processus (de PID 0) du système est le `swapper`. Ce processus lance le processus principal du système appelé `init` (de PID 1), qui lance lui-même les processus les plus importants du système. Tout ces processus sont organisés de façon arborescente. On dit en effet que `init` est le fils du `swapper` et inversément que le `swapper` est le père de `init`.

3.2 Commandes de consultation de l'états des processus

3.2.1 `ps`

La commande `ps` permet de consulter l'état des processus. Sans option, la commande `ps` affiche toute la liste des processus lancés à partir du *shell*. On peut entre autre y voir pour chaque processus, son PID, ainsi que l'utilisateur qui l'a lancé. Pour obtenir plus d'informations sur les processus, il faut utiliser l'option `-f` (*full*). Pour obtenir la liste de tout les processus du système, il faut utiliser l'option `-e`.

3.2.2 `pstree`

La commande `pstree` permet d'afficher toute l'arborescence des processus à partir d'`init`. On peut également passer à la commande `pstree` un PID. Dans ce cas, seul le sous-arbre dont la racine est le processus dont le PID est précisé en paramètre sera affiché.

3.2.3 `top`

La commande `top` permet d'afficher interactivement les processus du système. De plus, `top` affiche toute une série d'informations comme la charge du processeur et la taille de mémoire de chaque processus. Ce programme permet également de trier les processus suivant un certain nombre de critères. Pour plus de détails, taper `h` une fois que `top` est lancé.

3.3 Modifier l'état des processus

3.3.1 `kill`

La commande `kill` permet de tuer un processus en cours d'exécution en utilisant son PID. Pour tuer un processus, il faut donc d'abord obtenier sont PID (en utilisant `ps` ou `top`). Ensuite, il faut passer ce PID en paramètre à la commande `kill`.

3.3.2 `killall`

La commande `killall` permet de tuer tous les processus en cours d'exécution dont le nom correspond au paramètre. Ainsi, par exemple, la commande `killall xemacs` tuera toutes les instances d'`xemacs` en cours d'execution.

3.4 Exercices

1. Lancez votre éditeur de texte préféré (xemacs, gedit, nedit, ...).
2. Quel est le processus père de cet éditeur ?
3. Combien de place mémoire occupe cet éditeur ?
4. Quelle est la portion de processeur utilisée par cet éditeur ?
5. Tuez cet éditeur en utilisant la commande `kill`.
6. Relancez-en deux autres et tuez les en utilisant la commande `killall`.

4 `man` pages

Les `man` pages sont des pages de manuels reprenant une foultitude d'informations. Ces `man` pages sont découpée en sections numérotées. La section 1 porte sur la majorité des commandes du `shell`, la section 2 porte sur les *system calls* (cf. cours), et la section 3 porte quant à elle sur fonctions standard de C. Pour les consulter, vous pouvez utilisez la commande `man <sujet>`. `man` affiche alors la première page correspondante au `<sujet>` dans l'ordre des sections. Ainsi, par exemple, `man printf` vous affichera la page correspondant au programme `printf`, et non la fonction C `printf`. Pour remédier à cela vous pouvez précisez la section du manuel dans laquelle vous voulez chercher le `<sujet>` en paramère. Dès lors, pour afficher la `man` page correspondante à la fonction C `printf` vous devez utiliser la commande `man 3 printf`.