Cours UNIX

Chapitre 2

Concepts avancés

Les utilisateurs

Sur UNIX, chaque utilisateur est identifié par un nom et par un UID (*User IDentification*). Il peut appartenir à plusieurs groupes, eux-mêmes identifiés par un nom et par un GID (*Group IDentification*). Un de ces groupes est le *groupe principal*.

Chaque objet du système (fichiers et processus) appartient à un utilisateur et à un groupe. Chaque processus transmet à ses descendants ses caractéristiques utilisateur/groupe, et ses droits vis-à-vis des autres objets du système sont déterminés par rapport aux droits de l'utilisateur propriétaire et du groupe.

Il existe un utilisateur appelé Super Utilisateur qui à TOUS les droits sur TOUS les objets du système. Il à pour nom `root`, pour UID 0 et appartient au groupe dont de GID est 0.

Certains utilisateurs sont des utilisateurs *système*, c'est-à-dire qu'il ne sont pas utilisables par des personnes et ne peuvent pas ouvrir de session sur une console ou un terminal (par exemple: `daemon`, `operator`, `bin`, ...). Ils sont destinés à être propriétaires et à donner des droits à des processus systèmes (des processus qui ne sont pas exécutés par un utilisateur normal).

Un utilisateur normal doit *ouvrir une session* sur un terminal pour pouvoir travailler. Une fois connecté, le système exécute pour lui un processus appelé *shell* qui lui permet d'exécuter des commandes, ou bien il exécute un shell qui lui-même exécute un script qui va initialiser un environnement de travail graphique.

Chaque shell exécuté après une connexion est attaché à un *terminal logique*. La particularité d'UNIX est qu'un utilisateur peut ouvrir une session de multiples manières: depuis une console directement attachée au serveur, depuis un terminal texte attaché au port série, au port parallèle ou au réseau, depuis un terminal graphique attaché au réseau ou depuis une autre machine (UNIX ou non) à travers le réseau.

La commande `id` permet de connaître vos UID et GID:

```
# id
uid=1001(michelon) gid=1002(users) groups=1002(users), 0(wheel),
1004(webadmin)
```

Le système de fichiers

➔ Structure

Le système de fichiers UNIX est une arborescence contenant des fichiers et des répertoires, bien entendu, mais aussi des fichiers spéciaux, des liens, des sockets de communications, ...

On parle souvent de Système de Gestion de Fichiers (SGF) pour désigner ces concepts sous UNIX.

Chaque répertoire a un père unique, sauf le répertoire racine /.

Voici quelques uns des répertoires les plus rencontrés:

- /etc fichiers de configuration et d'administration du système
- /bin commandes utilisateur principales
- /sbin commandes système principales
- /dev fichiers spéciaux des périphériques
- /usr commandes et logiciels utilisateurs
- /var fichiers variables comme les logs ou les bases de données
- /home, /user répertoires des utilisateurs
- /tmp, /usr/tmp fichiers temporaires
- /lib, /usr/lib librairies partagées
- /usr/share fichiers de données partagés (comme les pages du manuel, la doc)

➔ Types de fichiers

Les types de fichiers les plus souvent rencontrés sont:

- Exécutables (binaires, scripts)
- Données
- Liens symboliques
- Périphériques
- Sockets
- Tubes nommés

En résumé, tous les objets sont considérés comme des fichiers. Souvenez-vous bien de cela,

c'est la base de la philosophie UNIX. Par exemple, pour jouer un morceau de musique, il suffit d'écrire les données du fichier son dans le fichier correspondant au périphérique carte son (comme `/dev/dsp` ou `/dev/snd`).

Autre particularité, un fichier peut avoir plusieurs noms (liens physiques) et des raccourcis (liens logiques).

➔ Les i-nodes

Un i-node ou noeud d'index est une zone sur le disque dans laquelle sont stockées les informations concernant un fichier. Voici les informations les plus importantes que l'on trouve:

- UID et GID du propriétaire
- Type du fichier
- Droits d'accès
- Taille en nombre d'octets
- Nombre de liens physiques (autrement dit, nombre de noms différents)
- Dates (lecture, modification,...)
- Adresses des blocs sur disque (si besoin est)

➔ Droits d'accès

Pour un fichier (ou un répertoire) donné, les droits peuvent être classés en trois catégories:

- le propriétaire du fichier (u)
- le groupe ayant accès au fichier (g)
- les autres (o)

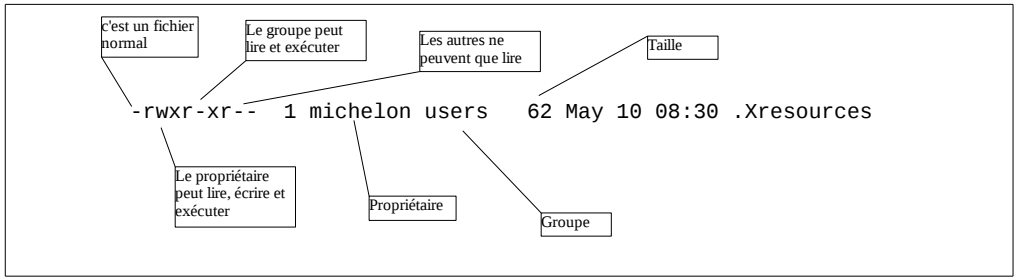
Les trois types d'opérations possibles sur les droits sont:

- la lecture (r)
- l'écriture (w)
- l'exécution (x)

La commande `ls` permet d'afficher les droits des fichiers et répertoires avec l'option `-l` (et donc d'afficher une partie des informations contenues dans l'i-node) :

```
# ls -l
drwxr-x---  3 root     wheel  512 Mar  8 07:17 mnt
-rwxr-xr--  1 michelon users   62 May 10 08:30 .Xresources
```

On peut voir ici que `mnt` est un répertoire (commence par 'd'), qu'il appartient au super utilisateur qui peut lire et exécuter ce répertoire (autrement dit, il peut utiliser la commande `cd` avec) et y créer des fichiers. Les utilisateurs du groupe wheel peuvent juste consulter le contenu de ce répertoire, et les autres utilisateurs n'y ont pas accès.



Les processus

➔ Généralités

Un processus est l'image en cours d'exécution d'un programme binaire. Du point de vue de la mémoire, un processus est constitué par:

- Le code compilé du programme
- Les données du programme
- Les informations système concernant le processus et son environnement d'exécution (le *PCB*, *Process Control Block*)

Sous UNIX, il existe deux types de processus:

- *Les processus systèmes*. Ce sont des processus qui ne sont pas attachés à un terminal d'entrée/sortie, et qui sont généralement exécutés sous l'UID du *Super Utilisateur* ou d'un *utilisateur système*. Ils sont aussi appelés démons (*daemons*).
- Les *processus utilisateurs*. Ils sont en général attachés au terminal depuis lequel ils ont étés exécutés, et ils le sont sous l'identité de l'utilisateur qui à tapé la commande.

Les commandes permettant de visualiser et de manipuler les processus sont: `ps`, `kill`, `killall`, `pkill`.

Chaque processus en cours d'exécution est identifié par un PID (Process IDentification) . Ce PID est unique pendant toute la durée de l'exécution mais peut être réutilisé une fois le processus terminé. Tous les processus en cours d'exécution ou de terminaison sont listés en mémoire dans la *table des processus*.

Tout comme le SGF (Système de Gestion de Fichiers) est hiérarchique, les processus sont aussi organisés de façon hiérarchisée: chacun d'entre eux à un seul et unique parent. On peut comparer la filiation des processus à l'organisation des répertoires dans le SGF.

Il existe donc un processus père de tous les autres, ayant comme PID 1. Ce processus est, dans la grande majorité des UNIX, le programme `init`, qui est chargé de lancer tous les processus systèmes (*daemons*) nécessaires au fonctionnement du système.

Souvent on peut voir aussi le processus qui a pour PID 0. Celui-ci n'est pas vraiment un processus mais c'est la représentation, dans la table des processus, de l'ordonnanceur des tâches du noyau (*scheduler*). C'est cet ordonnanceur qui à pour rôle de répartir les processus sur les processeurs et de gérer le partage du temps des processeurs entre tous les processus.

Voici quelques noms de processus systèmes souvent rencontrés sous UNIX:

- `init`. Le premier processus exécuté après le chargement du noyau et des pilotes. C'est lui qui initialise (d'où son nom) tous les processus système et qui lance les *shells* lors de la connexion d'un utilisateur.
- `syslogd`. Chargé de stocker (ou de transmettre) les messages des processus systèmes (qui ne peuvent pas communiquer avec l'extérieur car ils ne sont pas attachés à un terminal).
- `inetd`. Supervision et gestion des services réseaux utilisant le protocole IP (Internet Protocol). Aussi appelé "Internet Super Server".
- `timed`. Synchronisation de la date et de l'heure avec les autres machines du réseau local.
- `cron`. Lancement des processus périodiques ou programmés.
- `portmap`. Gestion des ports RPC.

Souvent les noms des processus systèmes parlent d'eux-même: `httpd` pour le serveur WEB HTTP, `named` pour le serveur de nom DNS, `nfsd` pour le système de fichier réseau NFS, ...

Il existe des commandes permettant de visualiser les processus en cours d'exécution: `ps` ou encore `top`. Voici un exemple de l'exécution de la commande `ps`:

```
# ps xl
UID   PID  PPID CPU PRI NI   VSZ  RSS WCHAN  STAT  TT       TIME COMMAND
  0     0     0   0 -18  0     0    0 sched  DLs   ??    0:00.68  (swapper)
  0     1     0   0  10  0   528  152 wait   ILs   ??    0:01.07 /sbin/init --
  0     2     0   0 -18  0     0    0 psleep DL    ??    3:05.19  (pagedaemon)
  0     3     0   0  18  0     0    0 psleep DL    ??    0:00.00  (vmdaemon)
  0    91     1   0   2 10   928  544 select SNs   ??    0:11.03 syslogd -s
  0    94     1   0   2 10  2952 1948 select SNs   ??    3:02.55 named
  0    97     1   0   2 10  1236  400 select INs   ??    0:02.10 timed
  0   101     1   0   2 10  1476 1088 select INs   ??    4:26.60 ypserv
  0   103     1   0   2 10  1072  684 select INs   ??    0:00.03 rpc.yppasswdd
  0   105     1   0   2 10   904  448 select INs   ??    0:08.16 ypbind
  0   113     1   1   2 10  1068  872 select INs   ??    0:01.82 mountd -r
  0   117   115   7  -6 10   352   68 biord  DN    ??   89:58.07 nfsd: server
  0   149     1   0  10 10   972  504 nanslp SNs   ??    0:10.67 cron
```

Note: `ps xl` fonctionne avec la version BSD de la commande. Sur un UNIX Système V, il faut utiliser `ps -el`.

On peut constater dans cet exemple la filiation des processus à l'aide des colonnes PID et PPID): le processus `nfsd` au PID 117 a pour père le processus `nfsd` au PID 115 qui lui-même a pour père le processus `init` au PID 1.

➔ Le PCB, Process Control Block

Comme les fichiers possèdent des caractéristiques stockées dans les i-nodes, à chaque processus est attaché un certains nombre d'informations stockés dans un PCB. Celui-ci est créé lors du chargement du processus en mémoire à l'aide des informations du processus père. Les informations contenues dans le PCB du processus père sont alors copiées dans le nouveau PCB.

Les caractéristiques d'un processus sont (attention: pas toutes ne sont stockées dans le PCB), entre autres (les noms des valeurs sont ceux utilisés sous UNIX en général et par les commandes `ps` et `top` en particulier):

- Identifiant unique: le PID
- Identifiant unique de son père: le PPID
- Identifiant de l'utilisateur auquel il appartient: l'UID
- Identifiant du groupe auquel il appartient: le GID
- Terminal (ou plutôt la voie de terminal) auquel il est attaché (si tel est le cas): le TTY
- Répertoire de travail, ou répertoire courant: le PWD (Process Working Directory)
- Priorité de ce processus par rapport aux autres: les valeurs PRI et NICE
- Temps cumulé d'exécution: la valeur TIME
- Date de lancement.
- Nom du fichier binaire correspondant: la valeur COMMAND.
- Liste des descripteurs de fichiers actuellement ouverts par le processus.
- État de l'exécution (valeurs des registres du processeur).

Ce sont ces informations qui sont affichées en partie par les commandes `ps` ou `top`.

➔ Terminaison, interruption

Un processus peut se terminer de lui-même, mais peut aussi être terminé à l'aide d'un signal, soit volontairement par un processus système ou par un utilisateur, soit par le système lors d'une erreur d'exécution.

Si le processus est attaché à un terminal, il peut être interrompu à l'aide du clavier par les touches CTRL-C ou CTRL-\. Ces combinaisons de touches envoient respectivement les signaux INT et QUIT.

On peut aussi envoyer des signaux à l'aides des commandes `kill`, `killall` ou `pkill` ou avec la fonction système `kill()`

Lors de sa terminaison, un processus passe dans un état spécial appelé *état zombi*. C'est le moment où le processus en cours de terminaison n'utilise plus aucune ressource du système et qu'il ne s'exécute plus, mais qu'il est encore en mémoire, et surtout, qu'il est encore listé dans la table des processus. Il peut arriver que certains de ces processus ne soient pas enlevés de la table des processus si leur terminaison n'a pas été prise en compte correctement par le processus père. Ce sont des zombies.

➔ Entrées/sorties

Lorsqu'un processus est attaché à un terminal, trois fichiers sont ouvert pour lui automatiquement:

- En lecture, l'entrée standard (*stdin*) (descripteur 0).
- En écriture, la sortie standard (*stdout*) (descripteur 1).
- En écriture, la sortie d'erreurs (*stderr*) (descripteur 2).

Physiquement, lorsqu'une commande est exécutée de façon interactive (depuis un shell), *stdin* correspond au clavier, *stdout* et *stderr* à l'écran ou à la fenêtre du terminal.

Dans UNIX, l'utilisation de ces entrées/sorties standards est très simple: elles sont considérées comme des fichiers, il suffit d'y lire et d'y écrire comme dans n'importe quel fichier (rappelez-vous que tout est considéré comme étant un fichier !).

Les processus peuvent donc communiquer entre eux par l'intermédiaire de ces entrées/sorties standards. Les commandes peuvent êtres liées entre elles de façon interactive lors de leur lancement simplement en redirigeant ou en enchaînant ces entrées/sorties.

Les redirections d'entrées/sorties sont des mécanismes offerts par les shells (voir chapitre 3, Les langages de commandes).

