# Gestion des utilisateurs & des processus sous Linux

## I. *Introduction à la gestion des utilisateurs*

Nous prenons l'exemple du système d'exploitation Linux pour s'initier aux tâches d'administration système. Un des atouts de ce système est sa facilité d'administration puisque la majorité des fichiers de configuration sont des fichiers textes pouvant être modifiés directement en utilisant un simple éditeur. Bien sûre il faut connaître quel fichier modifier? et comment? pour changer tel ou tel paramètre du système. La plupart des systèmes dérivés d'Unix présentent plus au moins la même hiérarchie des répertoires. Dans le tableau suivant nous citons les principaux répertoires du système Linux avec une brève description de leur contenu.

Le compte d'un utilisateur est représenté par le **login** et **un mot de passe** associé. Les informations sur les comptes utilisateurs disponibles sur une machine Unix sont regroupées dans le fichier **/etc/passwd**. Chaque ligne de ce fichier correspond à un compte. Une ligne est composée de **7 champs** séparés par des **: (deux points)**. Les champs sont les suivants :

| Champs | Contenu |
|---|---|
| Uname | c'est le login |
| Mot de passe | Il s'agit du mot de passe crypté. Quand le système **shadow password** est utilisé ce champ contient le caractère **x**. |
| UID | Numéro **unique** d'identification de l'utilisateur. |
| GID | Numéro d'identification du groupe principal auquel appartient l'utilisateur |
| Gecos | Ce champ est de type informatif. Il est à la disposition de l'administrateur qui l'utilise habituellement pour indiquer le nom et le prénom de l'utilisateur. |
| Homedir | répertoire personnel de l'utilisateur |
| Login shell | Shell initial exécuté à la connexion. |

Les informations concernant les groupes sont conservées dans le fichier **/etc/group**. Ce fichier contient une entrée par groupe. Chaque entrée est composée de **4 champs**:

| Champs | Contenu |
|---|---|
| Gname | Nom du groupe |
| Mot de passe | Rarement utilisé |
| GID | Identificateur du groupe |
| Liste des membres | c'est la liste des logins des utilisateurs membres du groupe. Les utilisateurs dont c'est le groupe principal n'ont pas besoin d'apparaître dans cette liste. |

Bien que les mots de passes soient sauvegardés d'une manière cryptée dans les fichiers **/etc/passwd** et **/etc/group**, des problèmes de sécurité se sont posés. En effet, ces deux fichiers doivent être en mode lecture pour tout le monde, ce qui implique que tous les utilisateurs peuvent avoir accès aux mots de passes, cryptés certes, des autres. Si le cryptage empêche une lisibilité directe, son efficacité n'est pas sûre à 100%. Il existe

aujourd'hui beaucoup des logiciels pour décoder les mots de passes cryptés. Seul des mots de passes longs et complexes nécessiteront un décryptage beaucoup plus long. La solution proposée par Linux (et d'autres système Unix) est de stocker les mots des passes dans un fichier où seul l'administrateur système a le droit de lire et de le modifier. Sous Linux, les mots de passe des utilisateurs sont stockés dans **/etc/shadow**. et ceux des groupes dans **/etc/gshadow**. En plus de sauvegarde des mots de passe, Linux offre avec le fichier **shadow** la possibilité de gérer un système de vieillissement des mots de passe. Autrement dit, l'administrateur peut fixer une **age limite** des mots de passe après laquelle l'utilisateur est invité à changer son mot de passe sous peine de ne plus avoir accès à son compte.

Une ligne dans le fichier **/etc/shadow** est composé de neuf champs séparés par le caractère **: (deux points)**.

| Champ | Contenu |
|---|---|
| Le login | |
| Mot de passe | Une * dans ce champ indique le compte ne peut être connecté (cas du compte bin par exemple). Un mot de passe commençant par !! indique que le compte est verrouillé. |
| Age | Le nombre de jour écoulé depuis le 1er janvier 1970 et la date de mise à jour du mot de passe. |
| Période de changement | Le nombre minimum de jours entre deux changements de mots de passe. Un 0 indique que l'utilisateur peut changer le mot de passe à n'importe quel moment. |
| Durée de validité | Le nombre maximum de jours pendant lesquels le mot de passe est valide. Le valeur 99999 indique que le mot de passe est toujours valide. |
| Durée de validité restant | Nombre de jours avant l'expiration. |
| Durée d'invalidation | Nombre de jour après l'expiration provoquant la désactivation du compte. Un champ vide indique qu'il n'y a aucune désactivation |
| Date d'expiration | Exprimée en nombre de jour depuis la date de référence (1/1/70) |
| Champs réservé | |

## *II. Compilation en plusieurs phases*

Le langage C est donc un langage compilé (même s'il existe des interpréteurs plus ou moins expérimentaux). Ainsi, un programme C est décrit par un fichier texte, appelé fichier source. La compilation se décompose en fait en 4 phases successives :

1) Traitement par le préprocesseur (passage au pré-processeur),
2) la compilation en langage assembleur,
3) transformation de l'assembleur en code machine, et
4) l'édition de liens.

Il existe évidemment un grand nombre de compilateurs C pour chaque plateforme (Windows, Linux, MAC etc..). Le compilateur utilisé dans les TPs est le compilateur libre **gcc (cc)**, disponible sur quasiment toutes les plateformes UNIX.

1) gcc -E NOM_MODULE.c > NOM_MODULE.i
2) gcc –S NOM_MODULE.i
3) gcc –C NOM_MODULE.S
4) gcc NOM_MODULE.O

**La command gcc NOM_MODULE.c ➔** compiler et faire l'édition de liens et la création de module exécutable d'une façon automatique et supprime les fichiers intermédiaires.
**Le command gcc –c NOM_MODULE.c ➔** faire la compilation et la création de fichier objet.
**Le command gcc NOM_MODULE.o ➔** faire l'édition de liens et la création de fichier exécutable.

## *III. Création d'un processus*

On appelle processus un objet dynamique correspondant à l'exécution d'un programme ou d'une commande GNU/Linux. Cet objet regroupe plusieurs informations, en particulier l'état d'avancement de chaque programme, l'ensemble des données qui lui sont propres, ainsi que d'autres informations sur son contexte d'exécution.

- Un processus possède des caractéristiques qui permettent au système de l'identifier. Parmi ces caractéristiques, on trouve en particulier :
- l'identifiant du processus (process ID – PID) qui est un nombre entier positif ;
- l'identifiant du processus qui lui a donné naissance, ou processus parent, appelé PPID (parent PID) ;
- l'identifiant de son propriétaire, en général (mais pas toujours) l'utilisateur qui l'a lancé ;
- son répertoire de travail ;
- sa priorité ;
- son temps d'exécution ;
- etc.

L'interpréteur de commandes « **shell** » est un programme et il devient un processus lorsqu'il est exécuté. Le programme shell par défaut des systèmes GNU/Linux est appelé « **bash** ». Chaque fois qu'une commande est exécutée dans un shell, un nouveau processus est créé à une exception : les commandes internes du shell (built-in shell), elles s'exécutent dans le contexte du shell. Il faut utiliser la commande **type** pour vérifier si une commande est une commande interne ou non.

Exemple : etudiant@Inf:~$ type cp ls which type

## *IV. Gestion des processus*

Il est possible de gérer les processus démarrés dans le système avec la commande **ps**. Pour afficher tous les processus du système, il faut taper :

etudiant@ Inf:~$ ps –Al

```
F S  UID  PID  PPID  C PRI NI ADDR  SZ  WCHAN  TTY   TIME       CMD
4 S    0    1     0  0  76  0  -   469    -     ?   00:00:00     init
1 S    0    2     1  0  94 19  -    0   ksofti ?   00:00:00 ksoftirqd/0
1 R 1000 4976  3574  0  76  0  -   7287   -     ?   00:00:01   konsole
```

Le programme **ps** affichera tous les processus actifs, leur PID, PPID, ainsi que d'autres informations. Options classiques de la commande **ps** :

- -A : afficher tous les processus.
- -l : format long.
- -a : afficher tous les processus, sauf les processus non rattachés à un terminal.
- -u [users] : afficher les processus appartenant à l'utilisateur ayant le nom spécifié.

Exemple :

Pour afficher tous les processus dans la session, taper :

etudiant@Inf:~$ ps -l

```
F S  UID   PID  PPID C PRI NI ADDR  SZ  WCHAN TTY    TIME     CMD
4 S 1000  5448  4981 0  77  0  -   639  wait  pts/1 00:00:00  su
0 S 1000  5450  5448 0  75  0  -   795  wait  pts/1 00:00:00  bash
0 R 1000  7050  5450 0  76  0  -   668   -    pts/1 00:00:00  ps
```

## V. Gestion des processus en temps réel

Pour gérer les processus en temps réel, il faut utiliser la commande top.

**etudiant@Inf:~$ top**

```
top - 15:13:13 up  3:53,  2 users,  load average: 1.30, 0.69, 0.53
Tasks: 219 total,   3 running, 216 sleeping,   0 stopped,   0 zombie
Cpu(s): 67.8%us, 20.5%sy,  0.0%ni, 11.4%id,  0.0%wa,  0.3%si,  0.0%st
Mem:   2572660k total,  2553380k used,    19280k free,   120660k buffers
Swap:  2120540k total,     8268k used,  2112272k free,   967508k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1189 root      20   0  312m 167m  66m R   58  6.7  21:53.07 Xorg
16792 etudiant  20   0  393m  22m 3032 S   56  0.9   0:01.72 thunderbird
12071 root      20   0  236m  49m  25m S   28  2.0   2:19.76 synaptic
16791 etudiant  20   0  393m  22m 2992 S   27  0.9   0:00.93 firefox
 1680 etudiant  20   0  338m 103m  64m S    3  4.1   8:20.80 compiz
 2008 etudiant  20   0  112m  19m 4340 S    2  0.8   0:37.28 beam.smp
 1894 etudiant  20   0  137m  30m 5588 S    1  1.2   0:29.50 ubuntuone
 3098 etudiant  20   0  935m 179m  98m S    1  7.1   2:05.15 soffice
16471 etudiant  20   0  329m  32m  16m S    1  1.3   0:01.81 terminal
    1 root      20   0 23828 1908 1288 S    0  0.1   0:00.74 init
```

Avec :

- %CPU : Le pourcentage du temps processeur utilisé par le processus.
- %MEM : Le pourcentage de la mémoire physique occupée par le processus.

Dès que top est lancée, il est possible d'exécuter des commandes interactives, taper :

- N : pour classer les processus par PID.
- A : pour classer les processus dans l'ordre chronologique (les plus récents en premier).
- P : pour classer les processus par rapport à leur utilisation CPU.
- M : pour classer les processus par rapport à l'utilisation de la mémoire.
- k : pour tuer un processus (le PID sera demandé).

## VI. Contrôle de l'exécution des processus

La commande **kill** permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas seulement à « tuer » un

processus. Il existe de nombreux signaux différents qu'on peut lister avec la commande **kill -l** (vous trouverez la signification dans le manuel : **man 7 signal**). Les signaux les plus courants sont :

- **SIGSTOP** (19) pour arrêter un processus.
- **SIGCONT** (18) pour continuer un processus arrêté.
- **SIGTERM** (15) pour signifier au processus qu'il doit se terminer.
- **SIGKILL** (9) pour tuer un processus.

La syntaxe d'envoi d'un signal est : kill –signal pid où signal est un numéro ou un nom de signal (le signal par défaut est SIGTERM).

Exemples :

```
etudiant@Inf:~$ kill -SIGKILL [PID]
etudiant@Inf:~$ kill -15 [PID]
```

## *VII. Contrôle des tâches*

Dans un processus **bash**, il est possible de démarrer plusieurs processus appelés aussi **jobs**. Par défaut, un processus est démarré en avant-plan et il est le seul à recevoir les données de l'entrée standard (le clavier). Il faut utiliser **Ctrl+Z** pour le suspendre. Pour démarrer un processus en arrière-plan, il faut utiliser le signe **&**.

**Exemples :**

```
etudiant@Inf:~$ gedit document.txt &
etudiant@Inf:~$ xeyes &
```

Les commandes pour manipuler les jobs sont les suivantes :

- Pour démarrer un processus en arrière-plan, il faut utiliser le caractère & :

```
etudiant@Inf:~$ xeyes &
[1] 12098
etudiant@Inf:~$
```

- Pour lister tous les jobs actifs :

```
etudiant@Inf:~$ jobs
[1]+ Running xeyes &
etudiant@Inf:~$
```

- Ramener un job en avant-plan :

```
etudiant@Inf:~$ fg %1
xeyes
```

- Suspendre un job, taper Ctrl+Z, le job passe dans un état arrêté :

```
[1]+ Stopped xeyes
etudiant@Inf:~$
```

- Continuer l'exécution d'un job tournant en arrière-plan :

```
etudiant@Inf:~$ jobs
[1]+ Stopped xeyes
etudiant@Inf:~$ bg %1
[1]+ xeyes &
etudiant@Inf:~$
```

- Tuer un job :

```
etudiant@Inf:~$ kill %1
```

Lorsque la session bash est terminée, tous les processus démarrés depuis ce shell reçoivent le signal **SIGHUP**, ce qui, par défaut, les stoppe. Dans le dernier exemple, le fait de tuer le processus bash tuera aussi le processus xeyes. Pour éviter qu'un processus ne s'arrête lorsque son parent se termine, le programme peut être lancé avec la commande **nohup** :

```
etudiant@Inf:~$ nohup xeyes
```