Cours Linux

1ère NSI - Architecture

qkzk

2022-07-19

Système d'exploitation & Linux

Système d'exploitation

Le seul langage que la machine comprenne est le langage machine. Il se programme en *assembleur* et ce n'est pas chose aisée. Il est donc nécessaire d'avoir un intermédiaire entre l'humain et la machine.

Le rôle principal d'un système d'exploitation (Operating System) est de traduire ce que nous voulons faire en langage machine.

Le système d'exploitation :

- fournit une interface entre l'humain et la machine
- gère les ressources de l'ordinateur (mémoire, processeur, périphériques, énergie . . .)
- gère les utilisateurs ainsi que leurs droits d'accès
- est indépendant du matériel
- rend concret ce qui ne l'est pas (un fichier est par essence abstrait mais nous le considérons bel et bien comme une entité concrète.)

Le système d'exploitation se sépare en deux grandes parties :

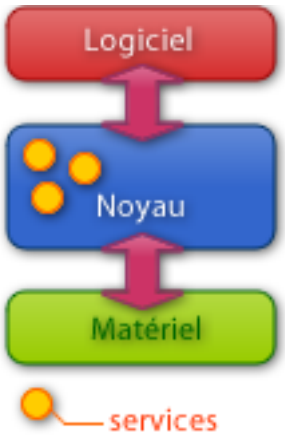

Figure 1: noyau

1. Le noyau (*kernel*) : gère les ressources de l'ordinateur et permet aux différents composants - matériels et logiciels - de communiquer entre eux. Le noyau n'est généralement pas accessible directement par l'utilisateur.
2. Les applications : utilisent l'interface proposée par le noyau et sont accessibles aux utilisateurs.

Les couches d'un OS

En étant un peu plus précis, on rencontre, du plus bas au plus haut niveau :

- Le matériel : (signaux électriques)
- Le noyau (kernel) : proche du métal. Lance la machine, gère la carte graphique, le réseau etc.
- La coquille (shell) : programme qui permet d'exécuter des utilitaires et d'intéragir (via les fenêtres ou le terminal)

- les applications (ls, firefox) : les programmes qu'on fait tourner grâce au shell

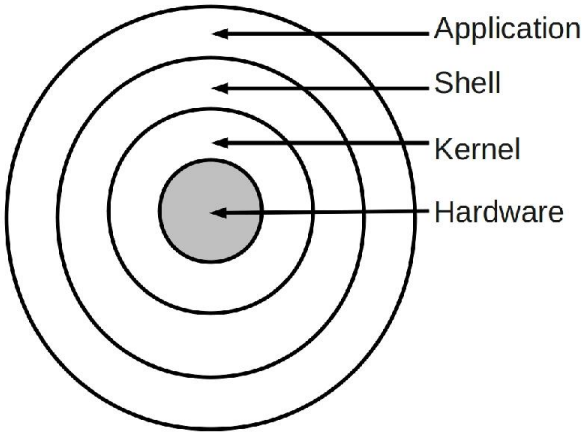

Figure 2: Couches d'un OS

Grandes familles de systèmes d'exploitation

On en rencontre massivement deux :

- **Windows** et ses dérivés (MSDOS (~1985), Windows NT (1999), windows 7->10 (2008)). Domine le marché du PC "personnel"
- **UNIX** et ses dérivés : bsd (systèmes embarqués, réseaux d'entreprises), linux (partout dont android, super calculateurs, PC personnels, serveur web), OSX & iOS (produits apple)

Linux et les systèmes libres

- Linux ou GNU/Linux est une famille de systèmes d'exploitation *open source* de type UNIX fondé sur le noyau Linux, crée en 1991 par Linus Torvald.
- Un système d'exploitiation de "type UNIX" vérifie quelques caractéristiques parmi lesqueles :
  - multi-utilisateur et multitâche
  - sécurisé vis-à-vis des manipulations illicites des utilisateurs
  - disposant d'un système de fichiers abouti
- UNIX était un système d'exploitation apparu dans les années 60, crée au Bell-Labs par Ken Thompson, Dennis Ritchie et Brian Kernighan.

  Les deux premiers sont les inventeurs du langage C qu'ils ont développé pour programmer UNIX.
- La majorité des "machines" modernes (téléphones, serveurs, super ordinateurs, informatique embarquée etc.) fonctionnent avec un système de type UNIX
- L'exception notable concerne les PC personnels et de bureau qui fonctionnent majoritairement sous windows.

GNU/Linux

Cet acronynime désigne :

- Le noyau linux lui même
- La couche logicielle GNU qu'on trouve dans tous les systèmes UNIX.

Ainsi, bien qu'il existe des centaines de distributions Linux, on retrouve les mêmes outils dans chacune et il n'est pas nécessaire de les connaître spécifiquement.

CLI, TUI, GUI

Concernant les applications, il existe trois types d'interface pour l'utilisateur :

Client Line Interface

Depuis un terminal, s'appuie sur un REPL (Read, Eval, Print, Loop)

```
$ ls -lah
Permissions Size User    Date Modified  Name
drwxr-xr-x     - quentin 19 mars   2021 cours-python
.rw-r--r--  157k quentin 27 sept.  2021 listes_projets.pdf
drwxr-xr-x     - quentin 15 janv. 07:40 premiere
.rw-r--r--    15 quentin  8 juil. 11:15 readme.md
drwxr-xr-x     - quentin 21 juin  10:25 terminale
```

On tape dans le console une commande, puis entrée, elle est lue, exécutée, la sortie apparait. Et on recommence.

C'est à la fois le plus simple et le plus puissant.

Terminal User Interface

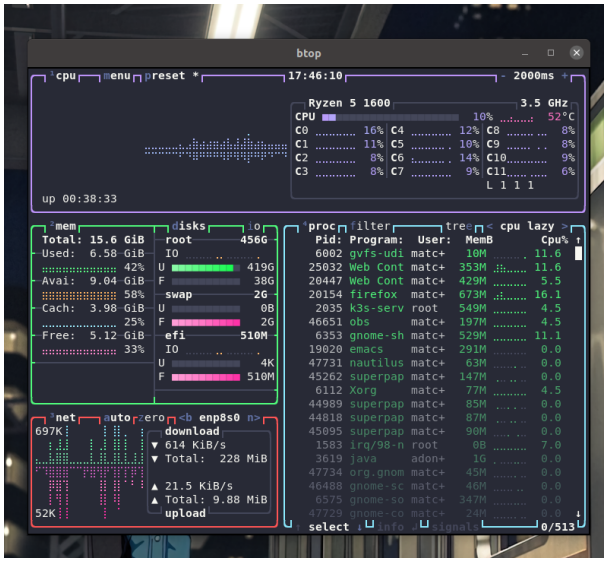

Figure 3: btop, une interface TUI

Toujours dans un terminal mais cette fois on peut intéragir (clavier, souris)

Graphical User Interface

Clic clic clic, inutile de présenter, vous connaissez.

Le terminal

Dans le vocabulaire courant shell, terminal et console désignent grosso modo la même chose : une fenêtre dans laquelle on peut taper des commandes.

Soyons plus précis :

- Shell : Un shell Unix est un interpréteur de commandes destiné aux systèmes d'exploitation Unix qui permet d'accéder aux fonctionnalités internes du système d'exploitation. Il se présente sous la forme d'une CLI accessible depuis la console ou un terminal
- Terminal : point d'accès de communication entre l'homme et un ordinateur central ou un réseau d'ordinateurs. il désigne par abus de langage une fenêtre d'invite de commande donnant accès à un shell Unix.
- Console : périphérique informatique de télécommunications des entrées-sorties d'un système de traitement de l'information. C'est généralement un terminal dédié uniquement à l'envoi et au retour des commandes.

Le shell est un *programme* comme bash ou zsh qui permet d'exécuter des commandes. C'est un langage de programmation interprété, un peu comme Python.

Le terminal, c'est le programme qui présente une fenêtre dans laquelle on envoie des commandes au shell.

Arborescence UNIX

UNIX voit ses périphériques et ses processus comme des fichiers.

Donc, en explorant certains dossiers on peut accéder aux périphériques ou aux processus. Les périphériques ne sont pas lisibles directement, de la même manière qu'une image nécessite un visionneur (sur le disque c'est des 0 et de 1), un périphérique nécessite *un pilote* pour être rendu accessible.

Dans un système UNIX, on dispose d'une **arborescence de fichiers** ancrée sur `/`, la "racine" (*root*). Voici quelques points d'entrée de cette arborescence :

```
/           ← root, la racine du système
+-- bin     ← Commandes de base du système
+-- dev     ← Fichiers représentant les dispositifs matériels (devices) du système
+-- etc     ← Fichiers de configuration du système
+-- home    ← Répertoire d'accueil (HOME) des utilisateurs
+-- lib     ← Librairies partagées entre les programmes
+-- mnt     ← Points de montage (clés usb etc.)
+-- proc    ← État du système et de ses processus
+-- root    ← Répertoire de l'administrateur système
+-- run     ← Variables d'état du système depuis le boot
+-- sys     ← Informations sur le noyau et les périphériques
+-- usr     ← Logiciels installés avec le système, base de données etc.
+-- var     ← Données fréquemment utilisées et modifiées
```

Les adresses des fichiers et dossiers sont séparées par des `/`

Par exemple : `/home/quentin/boulot/NSI/devoirs/DS1.pdf`

Navigation

Dans le shell, on se *situe* toujours dans un répertoire. Les commandes exécutées sont relatives à celui-ci.

On utilise généralement trois commandes pour naviguer :

- `cd` qui change de répertoire,
- `ls` qui affiche le contenu,
- `pwd` qui affiche l'adresse du répertoire courant.

Par ailleurs, chaque repertoire contient deux liens :

- `.` qui pointe vers lui même. N'est utilisé que pour lever des ambiguités dans les commandes,
- `..` qui pointe vers le parent.

Par exemple :

```
$ pwd
/home/toto/travail
$ ls
je_ne_veux_pas_travailler.txt
$ cd ..
$ pwd
/home/toto
$ ls
Documents travail repos vacances bonjour.py
$ python bonjour.py
Bonjour Toto !
```

Permissions

La sécurité sous unix est gérée par la notion de permission.

- Un utilisateur ne peut pas faire ce qu'il veut.
- Le super utilisteur `root` peut tout faire.
- Lorsqu'on est connecté à une machine on peut devenir un autre utilisateur avec `su`.

L'affichage détaillé d'un fichier (`ls -lah`) montre les permissions de

- l'utilisateur courant
- de son groupe
- et de tout le monde

Exemple :

```
$ ls -lah
-rwxr-xr-x 1 quentin quentin  324  2 déc.  21:45 deploy.sh
-rw-r--r-- 1 quentin quentin 3,6M  5 déc.  08:32 inside.log
```

De gauche à droite :

- `-rwxr-xr-x` : permissions
- `1` : nombre de référence à ce fichier
- `quentin quentin` propriétaire et groupe du propriétaire
- `324` : taille (par défaut en octet)
- `2 déc. 21:45` : date et heure de modification
- `deploy.sh` : nom du fichier

traduction des permissions

```
- : désactivé
d : directory
r : droit de lecture
w : droit d'écritude
x : droit d'exécution
```

- `deploy.sh -rwxr-xr-x` : ce n'est pas un dossier, je peux lire et écrire dans le fichier, l'exécuter. Mon groupe ne peut pas y écrire, les autres non plus.
- `inside.log -rw-r--r--` : tout le monde peut le lire, je suis le seul à pouvoir y écrire.

Modifier les permissions

On change les permissions avec `chmod`

- soit en ajoutant ou retirant un flag : `$ chmod +x inside.log` rendra ce fichier exécutable
- soit en décrivant la permission par un nombres à trois chiffres : `chmod 124 inside.log`
  - 1 : je peux exécuter
  - 2 : mon groupe peut écrire
  - 4 : tlm peut lire

  On fait la somme des nombres qu'on veut activer : Ex 1 + 2 + 4 = 7 = tous les droits.

  `$ chmod 764 inside.log` je peux tout faire, mon groupe ne peut pas exécuter, tlm peut lire.

Exercice :

1. Donner toutes permissions au fichier `recette.txt` du dossier courant
2. On a exécuté `chmod 644 travail/devoir.txt`. Quels sont les permissions du fichier ?

Les commandes de base du shell

Toutes ces commandes acceptent de nombreuses options dont on peut consulter la documentation en tapant `man ls` par exemple pour la commande `ls`. Celle-ci comporte des options pour affiche les fichiers cachés `ls -a` ou encore pour afficher les détails et permissions d'un fichier `ls -l`

Commande	Description
`cd`	*Change Directory* Se déplacer dans l'arborescence
`ls`	*Lister* le contenu du répertoire courant
`pwd`	*Affiche le dossier courant* (Print Workgin Directory)
`cp`	*Copier* des fichiers ou des répertoires

Commande	Description
`mv`	*Déplacer (move)* ou renommer des fichiers ou des répertoires
`rm`	*Effacer (remove)* des fichiers ou des répertoires
`cat`	*Visualiser (concaténer)* le contenu d'un fichier
`echo`	*Afficher* un message ou le contenu d'une variable
`touch`	*Créer* un fichier vide ou réinitialiser le *timestamp* d'un fichier
`ps`	Afficher les informations des *processus* en cours
`top`	*Gestionnaire de ressource* en TUI
`kill`	Envoyer un *signal* au processus, généralement pour l'arrêter
`grep`	*Filtrer* une sortie en ne gardant que les lignes contenant un terme
`nano`	*Éditeur de texte* en ligne de commande. Il y en a beaucoup

Redirection des entrées sorties

UNIX fonctionne principalement avec des petits programmes exécutant quelques tâches simples, le plus souvent une seule.

Ils communiquent avec des flux de texte qu'on peut enchaîner ou rediriger.

On peut, par exemple :

- écrire dans un fichier `$ cat cours.txt >> bonjour.txt` va recopier cours.txt dans bonjour.txt
- rediriger : `$ ps -ef` affiche 20 pages... `$ ps -ef | less` les fait défiler une par une !

En détail : `stdin, stdout, stderr`

Ce sont les trois flux de données crées lorsqu'on exécute une commande unix. Ces *flux* transfèrent des données, en l'occurence du texte.

Lorsqu'on tape une commande dans un terminal, l'entrée `stdin` est branchée sur le terminal. Sans paramètre supplémentaire, la sortie `stdout` est la fenêtre du terminal. La sortie d'erreur `stderr` est aussi la fenêtre.

Ces entrées et sorties peuvent être redirigés afin d'enchaîner les programmes.

Les redictions courantes sont :

- `>` qui renvoie la sortie standard vers un *fichier* `ls > fichier.txt` : écrit la sortie de `ls` dans le fichier.
- `|` qui branche la sortie standard d'un programme sur l'entrée standard du suivant `ps - ef | grep python` : parmi tous les processus (`ps -ef`), filtre `grep` ceux dont la description contient le mot `python`.
- `<` qui branche l'entrée standard sur le contenu d'un fichier. `wc -l < fichier.txt` compte les lignes du fichier.

Processus

- Un **programme** est un fichier (texte ou binaire) *que la machine peut exécuter.*
- Un **processus** est un *programme en cours d'exécution.*

Lors du lancement de l'OS, le noyau s'exécute d'abord. Il lance ensuite un processus applicatif (*init* ou *systemd*) ayant le numéro 1 qui va lancer tous les autres programmes.

Ce numéro, appelé "Processus IDentifier" ou **PID** est unique à chaque processus.

Le processus de PID 1 est donc *parent* de tous les autres.

- Lorsqu'on arrête un processus, tous ses enfants sont arrêtés aussi.
- Chaque processus peut créer d'autres processus enfants.

Par exemple, on ouvre un navigateur de PID 43256, chaque fois qu'on ouvre un nouvel onglet, un nouveau processus est crée avec un PID plus élevé.

Informations des processus

Elles sont toutes dans le dossier `/proc`... mais ce n'est pas très commode.

On accède aux processus avec `ps`

`ps -ef` permet d'afficher des colonnes :

```
UID     PID  PPID  C STIME      TTY      TIME    COMMAND
root    1    0     0 Dec 6      ?        1:02    init
...
jean    319 300    0 10:30:30   ?        0:02    /usr/dt/bin/dtsession
olivier 321 319    0 10:30:34   ttyp1    0:02    csh
olivier 324 321    0 10:32:12   ttyp1    0:00    ps -ef
```

Les colonnes de `ps -ef` sont :

La signification des différentes colonnes est la suivante :

- `UID` nom de l'utilisateur qui a lancé le process
- `PID` numéro du processus
- `PPID` numéro du processus parent
- `C` au facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
- `STIME` heure de lancement du processus
- `TTY` nom du terminal
- `TIME` durée de traitement du processus
- `COMMAND` commande ayant exécuté le nom du processus.

```
$ ps -ef | grep python
quentin  26945 10317  0 08:32 pts/1    00:00:00 /usr/bin/python -O /usr/bin/ranger
```

Affiche tous les processus (`ps`), dans une table (`-ef`) et filtre (`grep`) pour ne garder que ceux qui font référence à Python (`python`).

En gros, les programmes Python qui tournent sur la machine.

Je fais tourner un programme appelé `ranger` (gestionnaire de fichiers). Son numéro (PID) est 26945. Le numéro de son parent est 10317. Si ranger a planté et que je veux le tuer :

Envoyer un signal à un processus

Les processus écoutent l'arrivée de signaux venant de l'utilisateur ou de l'OS. Ces signaux sont transmis, par exemple, avec la commande `kill`.

```
$ kill 26945
```

Sans argument particulier, `kill` envoie un `SIGTERM`, en fait une constante valant 15.

Cela *demande poliment* au processus de s'arrêter, lui laissant le temps de terminer certaines opérations.

Donc, cela demande à `ranger` de s'arrêter.

Si, pour une raison quelconque, `ranger` ne s'arrête pas, on peut le contraindre avec le signal 9, pour `SIGQUIT` :

```
$ kill -9 26945
```

Hormis 9, tous les signaux peuvent être interprétés librement par le processus, c'est laissé à la liberté du développeur. Le bon usage est néanmoins de respecter la sémantique des signaux...