Administration Système sous Linux (Ubuntu Server)

Grégory Morel

2022-2023

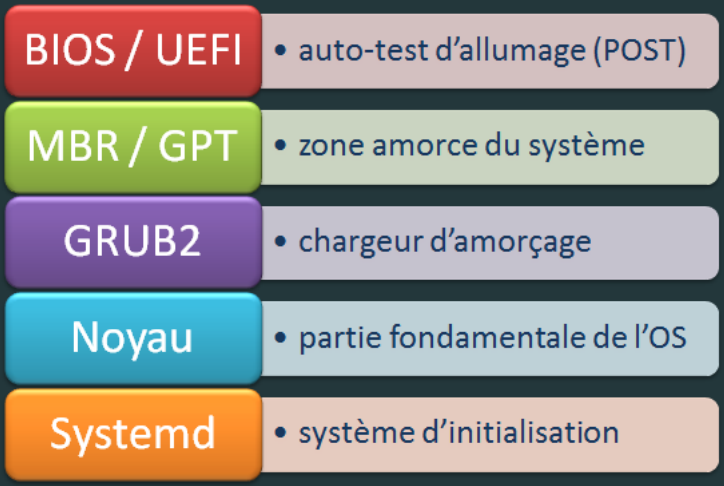
CPE Lyon - 3IRC / 4ETI / 3ICS

Cours 7

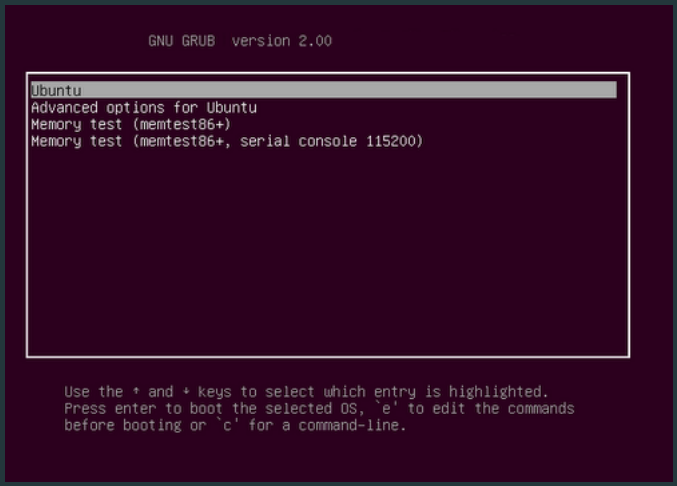
Boot, Noyau, services et processus

Boot

Séquence de démarrage d'un ordinateur (sous Ubuntu)



GRUB (*GRand Unified Bootloader*)



1. 💡 Si ce menu n'apparaît pas, il faut appuyer longuement sur la touche "Shift" ou "Echap" juste après le chargement du BIOS

GRUB (*GRand Unified Bootloader*)

GRUB2 : chargeur d'amorçage par défaut d'Ubuntu depuis la version 9.10 [1]

⇒ permet l'*amorçage* d'un système d'exploitation (Linux, mais aussi Windows)

⇒ propose un menu permettant de choisir parmi plusieurs OS, ou des options de récupération

⇒ paramétrable (modifier `/etc/default/grub` puis exécuter `update-grub`)

⇒ permet de passer des paramètres au noyau et au système d'initialisation

⇒ doc de GRUB : `info grub`

1. Sur d'autres systèmes, on peut trouver LILO (*Linux Loader*)

GRUB : fichier `/etc/default/grub`

Variables principales :

Variable GRUB	Signification
`GRUB_DEFAULT`	Entrée de menu sélectionnée par défaut
`GRUB_TIMEOUT_STYLE`	Options possibles : - `menu` : affiche le menu - `hidden` : le menu est caché - `countdown` : affiche un compte à rebours avant le boot par défaut
`GRUB_TIMEOUT`	Durée (en s) avant boot sur l'entrée par défaut
`GRUB_DISTRIBUTOR`	Texte de l'entrée du menu
`GRUB_CMDLINE_LINUX`	Paramètres passés au noyau Linux
`GRUB_GFXMODE`	Résolution utilisée par GRUB2 en mode graphique, qui peut être modifiée par une valeur retournée par la commande `vbeinfo`

GRUB : construction du menu

Les menus sont construits par les scripts présents dans `/etc/grub.d/` :

```
$ ll /etc/grub.d/
-rwxr-xr-x 1 root root 10364 oct.  17 20:44 00_header*
-rwxr-xr-x 1 root root  6258 oct.  17 20:41 05_debian_theme*
-rwxr-xr-x 1 root root 13716 oct.  17 20:44 10_linux*
-rwxr-xr-x 1 root root 11495 oct.  17 20:44 20_linux_xen*
-rwxr-xr-x 1 root root  1992 janv. 28  2016 20_memtest86+*
-rwxr-xr-x 1 root root 12059 oct.  17 20:44 30_os-prober*
-rwxr-xr-x 1 root root  1418 oct.  17 20:44 30_uefi-firmware*
-rwxr-xr-x 1 root root   214 oct.  17 20:44 40_custom*
-rwxr-xr-x 1 root root   216 oct.  17 20:44 41_custom*
-rw-r--r-- 1 root root   483 oct.  17 20:44 README
```

Ces fichiers sont traités dans l'ordre de listage.

Le fichier `10_linux` permet de rechercher les noyaux Linux

Le fichier `30_os-prober` permet de rechercher les autres OS

Noyau

Noyau Linux

Noyau (ou kernel) = coeur de GNU/Linux : gère les ressources, sert d'interface entre le matériel et les programmes

- développé par Linus Torvalds en 1991
- libre : on peut consulter le code source (`kernel.org`), le modifier, l'adapter...
- présent dans `/boot` et son nom commence par convention par `vmlinuz-X.Y.Z.p-V`[1]
- à l'origine monolithique : tous les composants et fonctionnalités sont regroupés dans un programme unique
- modulaire depuis la version 2.0 : certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée

1. Connaître la version utilisée : `uname -r` ou `cat /proc/version`

Noyau Linux

Noyau (ou kernel) = coeur de GNU/Linux : gère les ressources, sert d'interface entre le matériel et les programmes

- développé par Linus Torvalds en 1991
- libre : on peut consulter le code source (`kernel.org`), le modifier, l'adapter...
- présent dans `/boot` et son nom commence par convention par `vmlinuz-X.Y.Z.p-V`[1]
- à l'origine monolithique : tous les composants et fonctionnalités sont regroupés dans un programme unique
- modulaire depuis la version 2.0 : certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée

1. Connaître la version utilisée : `uname -r` ou `cat /proc/version`

Noyau Linux

Noyau (ou kernel) = coeur de GNU/Linux : gère les ressources, sert d'interface entre le matériel et les programmes

- développé par Linus Torvalds en 1991
- libre : on peut consulter le code source (`kernel.org`), le modifier, l'adapter...
- présent dans `/boot` et son nom commence par convention par `vmlinuz-X.Y.Z.p-V`[1]
- à l'origine monolithique : tous les composants et fonctionnalités sont regroupés dans un programme unique
- modulaire depuis la version 2.0 : certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée

1. Connaître la version utilisée : `uname -r` ou `cat /proc/version`

Noyau Linux

Noyau (ou kernel) = coeur de GNU/Linux : gère les ressources, sert d'interface entre le matériel et les programmes

- développé par Linus Torvalds en 1991
- libre : on peut consulter le code source (`kernel.org`), le modifier, l'adapter...
- présent dans `/boot` et son nom commence par convention par `vmlinuz-X.Y.Z.p-V`[1]
- à l'origine monolithique : tous les composants et fonctionnalités sont regroupés dans un programme unique
- modulaire depuis la version 2.0 : certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée

1. Connaître la version utilisée : `uname -r` ou `cat /proc/version`

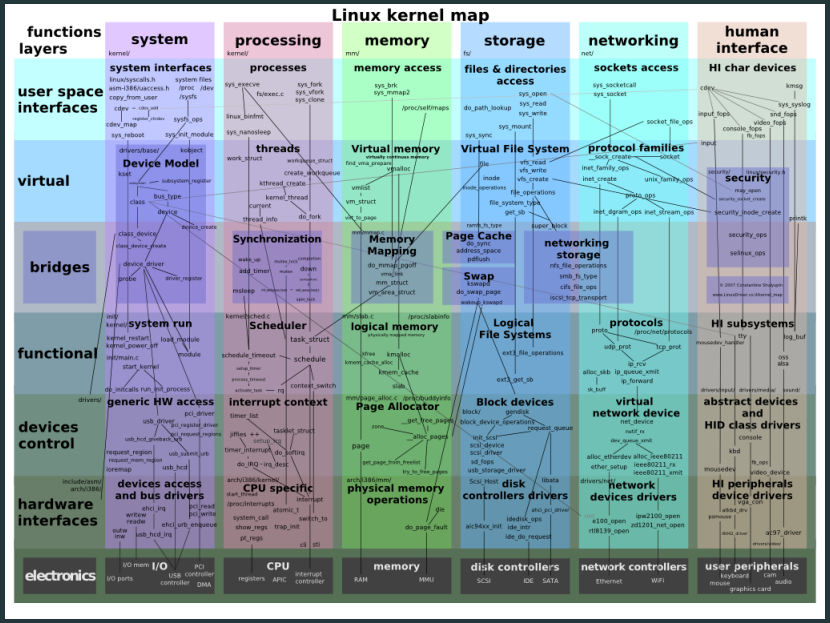
Noyau Linux

Noyau (ou kernel) = coeur de GNU/Linux : gère les ressources, sert d'interface entre le matériel et les programmes

- développé par Linus Torvalds en 1991
- libre : on peut consulter le code source (`kernel.org`), le modifier, l'adapter...
- présent dans `/boot` et son nom commence par convention par `vmlinuz-X.Y.Z.p-V`[1]
- à l'origine monolithique : tous les composants et fonctionnalités sont regroupés dans un programme unique
- modulaire depuis la version 2.0 : certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée

1. Connaître la version utilisée : `uname -r` ou `cat /proc/version`

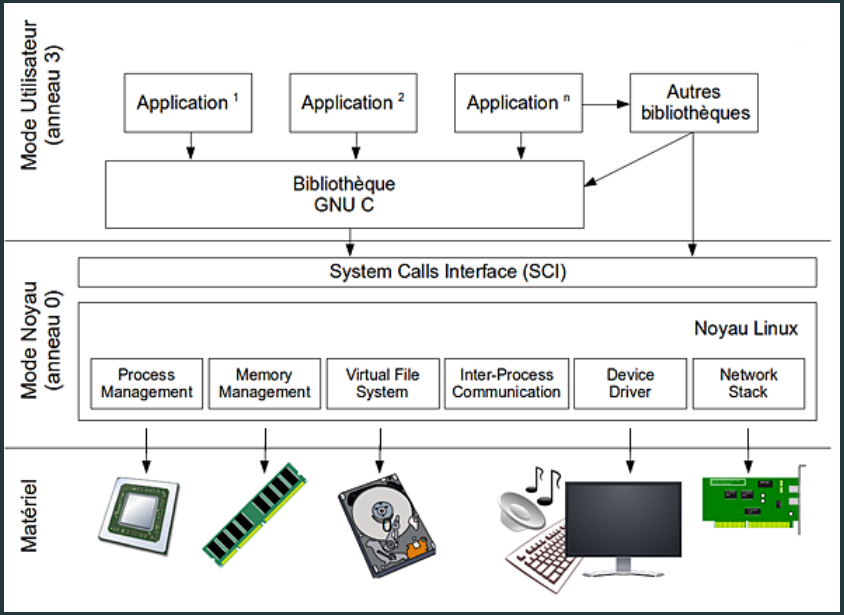
Noyau Linux


Linux kernel map
functions layers
system
processing
memory
storage
networking
human interface
user space interfaces
virtual
bridges
functional
devices control
hardware interfaces
electronics
system interfaces
processes
memory access
files & directories access
sockets access
HI char devices
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
Synchronization
Memory Mapping
Page Cache
networking storage
Swap
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
I/O
CPU
memory
disk controllers
network controllers
user peripherals

6 sous-systèmes principaux :

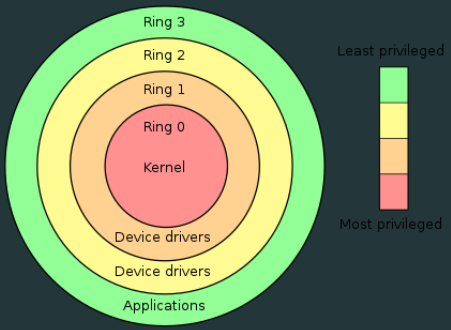
Sous-système	Rôle
Process Management (PM)	Ordonnancement des tâches
Memory Management (MM)	A chaque processus sa zone mémoire
Inter-Process Communication (IPC)	Permet aux applications de communiquer entre elles
Virtual File System (VFS)	Gestion correcte des fichiers / contrôle des droits d'accès ; encapsule le système de fichiers réel
Device Driver (DD)	Gère les ressources matérielles et fournit une interface uniforme pour l'accès à ces ressources
Network Stack (NET)	Gestion du réseau

Noyau Linux


Mode Utilisateur (anneau 3)
Application 1
Application 2
Application n
Autres bibliothèques
Bibliothèque GNU C
System Calls Interface (SCI)
Mode Noyau (anneau 0)
Noyau Linux
Process Management
Memory Management
Virtual File System
Inter-Process Communication
Device Driver
Network Stack
Matériel

Anneaux de protection

Anneaux de protection = niveaux de privilège[1]



L'organisation des anneaux de protection varie si l'installation est physique ou virtualisée, et entre 32 et 64 bits.

1. Apparus avec Multics, l'ancêtre d'UNIX !

Anneaux de protection

Gérés par le processeur, pas par l'OS [1]

? Comment un processus utilisateur réalise-t-il un appel système ?
⇒ en réalisant une commutation de contexte :

1. le contexte du processus courant est sauvegardé
2. le processeur bascule en mode noyau
3. l'appel système est exécuté
4. le processeur revient en mode utilisateur
5. le contexte du processus appelant est restauré

1. Sur les compatibles x86, on peut connaître le niveau de privilège d'une instruction en regardant les bits `IOPL` du registre `RFLAGS`.

Modules

Modules

Les composants de base (gestion de la mémoire, des processus...) sont regroupés dans un unique programme (le "coeur du kernel").

Depuis la version 2.0, le kernel contient aussi des modules ou LKM [1] qui peuvent être chargés ou déchargés à chaud : pilotes, systèmes de fichiers, pare-feu, protocoles réseau...

Les modules disponibles sont présents dans `/lib/modules` et ont une extension `.ko` (*kernel object*).

1. Loadable Kernel Modules

Modules

La commande `lsmod` liste les modules chargés[1], avec leurs dépendances :

```
$ lsmod
Module                  Size  Used by
iptable_filter         16384  0
bpfilter               16384  0
vboxsf                 45056  0
snd_intel8x0           40960  2
snd_ac97_codec        131072  1 snd_intel8x0
```

Gestion des modules :

`modprobe -a`	charge le module et ses dépendances
`modprobe -r`	décharge le module
`insmod`	charge le module sans ses dépendances
`rmmod`	décharge le module

1. On peut aussi regarder le fichier `/proc/modules`

Systemd

Système d'initialisation

- Premier programme exécuté et dernier arrêté
- Historiquement, sur les OS dérivés d'UNIX, il s'agit de `init` ou `SysVinit`
- Ubuntu 6.10 : introduction d'`Upstart`, plus souple
- Parallèlement, un autre projet est développé : `systemd` ⇒ système d'initialisation par défaut de la plupart des distributions (Ubuntu depuis 15.04)[1]

Une bonne infographie sur les différences entre SysVinit et Systemd :
`http://images.linoxide.com/systemd-vs-sysVinit-cheatsheet.pdf`

1. `init` existe encore sous Ubuntu ; mais c'est un simple lien sur `systemd`

Système d'initialisation

- Premier programme exécuté et dernier arrêté
- Historiquement, sur les OS dérivés d'UNIX, il s'agit de `init` ou `SysVinit`
- Ubuntu 6.10 : introduction d'`Upstart`, plus souple
- Parallèlement, un autre projet est développé : `systemd` ⇒ système d'initialisation par défaut de la plupart des distributions (Ubuntu depuis 15.04)[1]

Une bonne infographie sur les différences entre SysVinit et Systemd :
`http://images.linoxide.com/systemd-vs-sysVinit-cheatsheet.pdf`

1. `init` existe encore sous Ubuntu ; mais c'est un simple lien sur `systemd`

Système d'initialisation

- Premier programme exécuté et dernier arrêté
- Historiquement, sur les OS dérivés d'UNIX, il s'agit de `init` ou `SysVinit`
- Ubuntu 6.10 : introduction d'`Upstart`, plus souple
- Parallèlement, un autre projet est développé : `systemd` ⇒ système d'initialisation par défaut de la plupart des distributions (Ubuntu depuis 15.04)[1]

Une bonne infographie sur les différences entre SysVinit et Systemd :
`http://images.linoxide.com/systemd-vs-sysVinit-cheatsheet.pdf`

1. `init` existe encore sous Ubuntu ; mais c'est un simple lien sur `systemd`

Système d'initialisation

- Premier programme exécuté et dernier arrêté
- Historiquement, sur les OS dérivés d'UNIX, il s'agit de `init` ou `SysVinit`
- Ubuntu 6.10 : introduction d'`Upstart`, plus souple
- Parallèlement, un autre projet est développé : `systemd` ⇒ système d'initialisation par défaut de la plupart des distributions (Ubuntu depuis 15.04)[1]

Une bonne infographie sur les différences entre SysVinit et Systemd :
`http://images.linoxide.com/systemd-vs-sysVinit-cheatsheet.pdf`

1. `init` existe encore sous Ubuntu ; mais c'est un simple lien sur `systemd`

Système d'initialisation

- Premier programme exécuté et dernier arrêté
- Historiquement, sur les OS dérivés d'UNIX, il s'agit de `init` ou `SysVinit`
- Ubuntu 6.10 : introduction d'`Upstart`, plus souple
- Parallèlement, un autre projet est développé : `systemd` ⇒ système d'initialisation par défaut de la plupart des distributions (Ubuntu depuis 15.04)[1]

Une bonne infographie sur les différences entre SysVinit et Systemd :
`http://images.linoxide.com/systemd-vs-sysVinit-cheatsheet.pdf`

1. `init` existe encore sous Ubuntu ; mais c'est un simple lien sur `systemd`

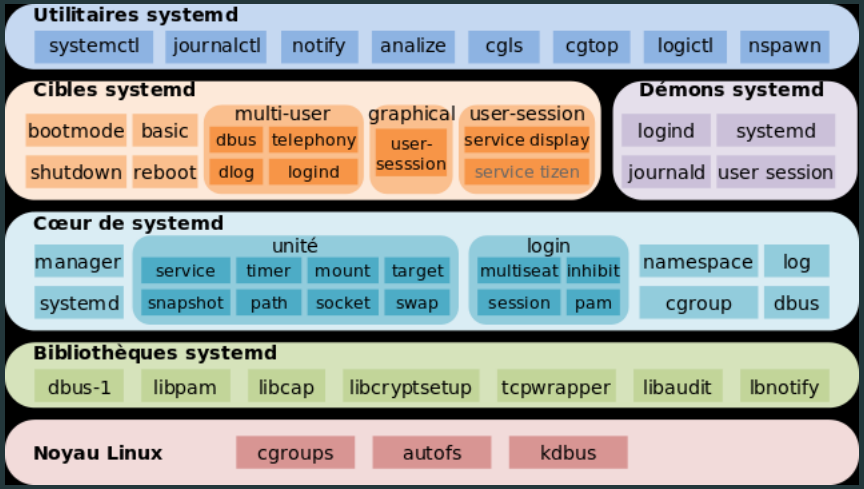
Systemd

Avantages :

- fonctionnalités de parallélisation ⇒ accélère le démarrage du système
- allocation fine des ressources (mémoire, processeur, E/S…)
- journaux systèmes plus complets (car démarrant plus tôt)
- séparations des services fournis par la distribution et ceux créés par l'administrateur
- gère le swap
- gère les journaux système
- gère les périphériques
- permet de monter ou démonter des points de montage
- …

Systemd

Systemd abandonne les scripts Shell et est basé sur environ 80 *binaires* :



Principaux utilitaires Systemd :

- `systemctl`[1] : gestion des unités
- `journalctl` : consultation des logs
- `timedatectl` : contrôle de l'horloge et du calendrier
- `machinectl` : gestion des conteneurs lancés par systemd
- `hostnamectl` : gestion du nom de la machine
- ...

💡 Depuis le passage à `systemd`, toutes les commandes `poweroff`, `halt`, `reboot`... sont des alias pour `systemctl`

1. A ne pas confondre avec `sysctl` qui permet de modifier certains paramètres du noyau "à chaud"

La configuration de Systemd se base sur des unités

Définition

Une unité désigne tout type de composant du système (service, périphérique, point de montage, timers...) géré par Systemd

Chaque unité utilise un nom d'extension en rapport avec son type :

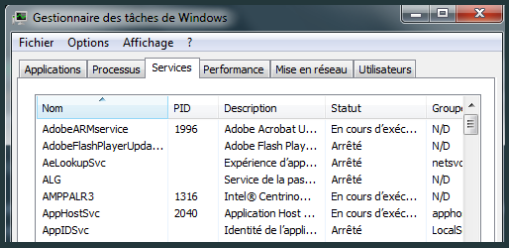
- service : `*.service`
- groupe d'unités : `*.target`
- point de montage : `*.mount`
- socket de communication inter-processus : `*.socket`
- ...

Services

Définition

Service ou daemon [1] : processus qui s'exécute en arrière-plan (plutôt que sous le contrôle direct de l'utilisateur) et réagit à des sollicitations extérieures

Pas propre à Linux :



Les serveurs (web, messagerie, base de données…) doivent fonctionner en permanence et sont souvent des daemons

1. D'où *systemd*, *apached*, *httpd*…

Exemple d'unité service :

```
/lib/systemd/system$ cat bind9.service
[Unit]
Description=BIND Domain Name Server
Documentation=man:named(8)
After=network.target
Wants=nss-lookup.target
Before=nss-lookup.target

[Service]
Type=forking
EnvironmentFile=-/etc/default/bind9
ExecStart=/usr/sbin/named $OPTIONS
ExecReload=/usr/sbin/rndc reload
ExecStop=/usr/sbin/rndc stop

[Install]
WantedBy=multi-user.target
```

Gestion des services

L'outil de gestion des services est `systemctl` :

```
systemctl ACTION <nom du service>
```

où `ACTION` est la commande à appliquer :

- `start` : démarre le service
- `stop` : arrête le service
- `restart` : relance le service
- `reload` : recharge la configuration du service
- `is-active` : affiche l'état (succinct) du service
- `status` : affiche l'état (détaillé) du service
- `enable` : active le lancement automatique du service au démarrage
- `disable` : désactive le lancement automatique du service au démarrage

Lister les services actifs :

```
systemctl list-units --type=service
```

Processus

Qu'est-ce qu'un processus ?

Définition

Un processus représente un programme en cours d'exécution, et son environnement d'exécution (mémoire, état, propriétaire, père...)

Plusieurs commandes permettent d'afficher la liste des processus :

- `ps` : affiche les processus lancés par l'utilisateur courant
- `ps -u user` : affiche les processus lancés par *user*
- `ps aux` : affiche tous les processus et par qui ils ont été lancés
- `top` : affiche une vue dynamique (`ps` n'affiche qu'un instantané) et propose plusieurs commandes pour trier ou gérer les processus [1]
- `htop` : version plus complète et conviviale de `top`, avec la charge mémoire et CPU

1. Pour les voir, appuyez sur la touche 'h' après avoir lancé *top*

Process ID

Chaque processus est repéré par un identifiant ou PID (*Process ID*)

Le premier processus lancé par le système, `init`, porte toujours le numéro `1`

Pour récupérer le PID d'un processus connaissant son nom :

```
$ pidof firefox (ou pgrep firefox)
3289 3227 3171 3064
```

Un processus ne peut pas naître spontanément : il naît toujours d'un autre processus, appelé processus parent et repéré par son PPID (`Parent Process ID`), par un processus appelé `fork`

```
$ ps -ef | grep mon_programme
greg    2468 2322 [...] mon_programme
```

Process ID

Chaque processus est repéré par un identifiant ou PID (*Process ID*)

Le premier processus lancé par le système, `init`, porte toujours le numéro `1`

Pour récupérer le PID d'un processus connaissant son nom :

```
$ pidof firefox (ou pgrep firefox)
3289 3227 3171 3064
```

Un processus ne peut pas naître spontanément : il naît toujours d'un autre processus, appelé processus parent et repéré par son PPID (`Parent Process ID`), par un processus appelé `fork`

```
$ ps -ef | grep mon_programme
greg    2468 2322 [...] mon_programme
```

Process ID

Chaque processus est repéré par un identifiant ou PID (*Process ID*)

Le premier processus lancé par le système, `init`, porte toujours le numéro `1`

Pour récupérer le PID d'un processus connaissant son nom :

```
$ pidof firefox (ou pgrep firefox)
3289 3227 3171 3064
```

Un processus ne peut pas naître spontanément : il naît toujours d'un autre processus, appelé processus parent et repéré par son PPID (`Parent Process ID`), par un processus appelé `fork`

```
$ ps -ef | grep mon_programme
greg    2468 2322 [...] mon_programme
```

Lancer un processus en arrière-plan

Même dans une console, on peut exécuter plusieurs tâches en même temps[1] ! On peut placer des programmes en arrière-plan pendant qu'on travaille sur un autre

Pour cela, on fait suivre la commande d'une *esperluette* & :

```
$ cp original.avi copie.avi &
[3] 1234
```

Le 3 signifie que c'est le troisième processus en arrière-plan dans cette console ; 1234 est le PID du processus

Un processus en arrière-plan continue d'écrire sur la sortie standard ⇒ rediriger la sortie ou utiliser la commande nohup

1. Les commandes ps ou systemctl l'ont prouvé !

Lancer un processus en arrière-plan

On peut aussi placer en arrière-plan un programme déjà en cours d'exécution !

Pour cela, on commence par mettre le programme en pause avec CTRL+Z, puis on tape bg (*background*) :

```
$ cp original.avi copie.avi
^Z
[3]+ Stopped        cp original.avi copie.avi
$  bg
[3]+ cp original.avi copie.avi &
$ jobs [3]+ Running        cp original.avi copie.avi &
```

Récupérer un processus en arrière-plan

La commande `jobs` affiche la liste des tâches en arrière-plan, et leur état[1] :

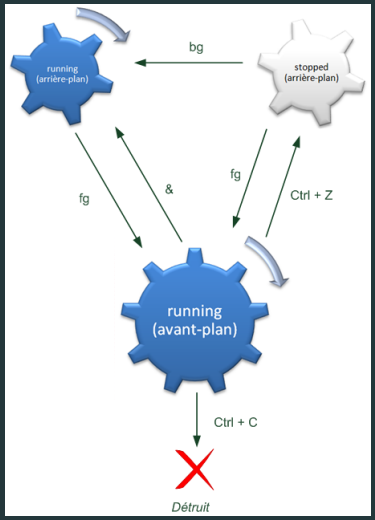
```
$ jobs
[1]-  Stopped      top
[2]   Running      firefox &
[3]+  Running      cp original.avi copie.avi &
```

On peut alors choisir de ramener un processus au premier plan avec

`fg %<numéro>`

1. Le `+` indique la dernière tâche mise en arrière-plan, et celle qui sera remise au premier plan avec `fg`. Le `-` indique l'avant-dernière

En résumé :



. Illustration : www.openclassrooms.com

Arrêter un processus en cours d'exécution

On utilise la commande `kill`

Attention !

Faux-ami ! `kill` sert avant tout à envoyer un signal à un processus

Pour arrêter un processus, on indique son PID :

```
$ ps aux | grep firefox
greg    2468 [...] 16:22    0:06 /usr/lib/firefox/firefox
$ kill 2468
```

La commande `kill` envoie par défaut le signal `SIGTERM`. Quand un processus est *vraiment* bloqué, on peut le *tuer* brutalement, en lui envoyant le signal `SIGKILL` (n° 9 sur la plupart des systèmes [1]) :

```
$ kill -SIGKILL 2468
```

1. Pour avoir la liste de tous les signaux : `kill -l`

Arrêter un processus en cours d'exécution

On utilise la commande `kill`

Attention !

Faux-ami ! `kill` sert avant tout à envoyer un signal à un processus

Pour arrêter un processus, on indique son PID :

```
$ ps aux | grep firefox
greg    2468 [...] 16:22    0:06 /usr/lib/firefox/firefox
$ kill 2468
```

La commande `kill` envoie par défaut le signal `SIGTERM`. Quand un processus est *vraiment* bloqué, on peut le *tuer* brutalement, en lui envoyant le signal `SIGKILL` (n° 9 sur la plupart des systèmes [1]) :

```
$ kill -SIGKILL 2468
```

1. Pour avoir la liste de tous les signaux : `kill -l`

Arrêter un processus en cours d'exécution

On utilise la commande `kill`

Attention !

Faux-ami ! `kill` sert avant tout à envoyer un signal à un processus

Pour arrêter un processus, on indique son PID :

```
$ ps aux | grep firefox
greg    2468 [...] 16:22   0:06 /usr/lib/firefox/firefox
$ kill 2468
```

La commande `kill` envoie par défaut le signal `SIGTERM`. Quand un processus est *vraiment* bloqué, on peut le *tuer* brutalement, en lui envoyant le signal `SIGKILL` (n° 9 sur la plupart des systèmes [1]) :

```
$ kill -SIGKILL 2468
```

1. Pour avoir la liste de tous les signaux : `kill -l`

Arrêter un processus en cours d'exécution

On utilise la commande `kill`

Attention !

Faux-ami ! **`kill`** sert avant tout à envoyer un signal à un processus

Pour arrêter un processus, on indique son PID :

```
$ ps aux | grep firefox
greg    2468 [...] 16:22   0:06 /usr/lib/firefox/firefox
$ kill 2468
```

La commande `kill` envoie par défaut le signal `SIGTERM`. Quand un processus est *vraiment* bloqué, on peut le *tuer* brutalement, en lui envoyant le signal `SIGKILL` (`n° 9` sur la plupart des systèmes[1]) :

```
$ kill -SIGKILL 2468
```

1. Pour avoir la liste de tous les signaux : `kill -l`

Arrêter un processus en cours d'exécution

La commande `killall` fonctionne comme `kill` mais permet de spécifier un processus par son nom :

```
$ killall firefox
```

La différence avec `kill` est que `killall` tue toutes les instances d'un programme. Par exemple, chaque fenêtre (pas *onglet*) Firefox est vue comme une instance distincte ; l'utilisation de `killall` ne fera pas dans le détail et tuera *toutes* les fenêtres

Important !

Un processus dont le père est tué est automatiquement adopté par le processus `init`

Arrêter un processus en cours d'exécution

La commande `killall` fonctionne comme `kill` mais permet de spécifier un processus par son nom :

```
$ killall firefox
```

La différence avec `kill` est que `killall` tue toutes les instances d'un programme. Par exemple, chaque fenêtre (pas *onglet*) Firefox est vue comme une instance distincte ; l'utilisation de `killall` ne fera pas dans le détail et tuera *toutes* les fenêtres

Important !

Un processus dont le père est tué est automatiquement adopté par le processus `init`

Arrêter un processus en cours d'exécution

La commande `killall` fonctionne comme `kill` mais permet de spécifier un processus par son nom :

```
$ killall firefox
```

La différence avec `kill` est que `killall` tue toutes les instances d'un programme. Par exemple, chaque fenêtre (pas *onglet*) Firefox est vue comme une instance distincte ; l'utilisation de `killall` ne fera pas dans le détail et tuera *toutes* les fenêtres

Important!

Un processus dont le père est tué est automatiquement adopté par le processus `init`