Processus

Séquence de boot

- BIOS : Basic Input/Output System (dont POST : Power-On Self-Test)
- Bootloader (MBR|UEFI-GPT, Grub) | PXE
- kernel (noyau Linux)
- initramfs (INITial RAM FileSystem) : décompression de l'arborescence temporaire `initrd` en RAM et exécution de l'exécutable `/init`
  - objectif : être en mesure de monter le fs qui sera monté sur `/` de façon permanente
  - pas forcément trivial : imaginez une machine accessible uniquement à distance dont les partitions sont chiffrées : l'initramfs doit contenir un micro-serveur `ssh` et l'exécutable `cryptsetup`. C'est aussi le moment où l'intégrité du fs qui sera monté sur `/` peut être verifié.
- `init` : le scheduler du noyau exécute le `/usr/sbin/init` de l'arborescence permanente. Dans la plupart des systèmes GNU/Linux, il s'agit de `systemd` (pour macos, c'est `launchd`).
- à partir de là se déploie l'arborescence des processus (forks depuis init)

Arborescence des processus

PID, PPID, PGID, arborescence, scheduler, init (user space), [kthreadd] (kernel space)

- observer l'arborescence des processus :

  ```
  pstree
  ps [--forest] -eo pid,ppid,pgid,uid,gid,stat,comm[and]
  htop
  ```

- daemons (ppid=1) :

  ```
  ps -eo ppid,comm | grep '^ *1 '
  ```

- les classiques : `ps aux` (BSD), `ps -eF` (POSIX)
- retrouver le PID d'après le nom de commande : `pidof`
- quels processus accèdent à quels fichiers : `fuser lsof`
- informations contenues dans `/proc` : `man 5 proc`

Systemd

`systemd` est le système d'initialisation de la plupart des distributions récentes. C'est ce démon (en:daemon = divinité : un processus qui tourne en arrière-plan plutôt que sous le contrôle direct d'un utilisateur) qui est lancé comme premier processus (`PID 1`) par le noyau et qui va se charger de gérer ses descendants.

L'adoption de `systemd` a suscité de nombreux débats dans la communauté GNU/Linux, car il s'arroge de nombreuses prérogatives au delà de l'init classique (gestion des logs, des périphériques, du réseau, etc), en contradiction avec un des points de la philosophie UNIX promouvant une architecture faite de petits programmes indépendants, simples, en interaction.

`systemd` gère des "unités" (en: units), la commande: `systemctl list-units` permet de voir l'étendue des prérogatives de `systemd` : devices, sockets, services... les unités qui nous intéressent sont les `services` (et les `targets` qui regroupent plusieurs services).

Systemd

Néanmoins, `systemd` offre de nombreux avantages sur son prédécesseur `sysVinit` :

- lancement parallèle des services grâce à une astuce d'utilisation de sockets
- on passe d'une description impérative du lancement des services (des scripts shell) à une description déclarative, avec une syntaxe unifiée (le répertoire `/etc/systemd/` contient des liens symboliques vers des fichiers de `/lib/systemd/system`, consultez-en quelques uns).
- des commandes agréables à utiliser (cf page suivante)
- l'utilisation de cgroup permettant un groupement arborescent des procéssus (au delà du `PGID` qui ne fait qu'une partition), voir `systemd-cgls`.

Systemd

La commande de `systemd` permettant de gérer les services est `systemctl`.

- Démarrer/éteindre un service :

```
systemctl start <service>
systemctl stop <service>
systemctl restart <service>
```

- État d'un service, et dernières lignes de son fichier de log :

```
systemctl status <service>
```

- Activer/désactiver le démarrage automatique d'un service :

```
systemctl enable <service>
systemctl disable <service>
systemctl is-enabled <service>
```

- Recharger la configuration d'un service :

```
systemctl reload <service>
```

Systemd

Cette commande peut aussi servir à changer l'état de la machine :

```
systemctl poweroff
systemctl reboot
systemctl suspend
systemctl hibernate
```

`systemd` fournit de nombreuses autres commandes de contrôle de divers aspects du système :

- `journalctl` (logs)
- `loginctl` (login)
- `hostnamectl` (nom d'hôte)
- `networkctl` (réseau)
- `resolvectl` (résolution DNS)
- `localectl` (paramètres régionaux (en: locale) et disposition du clavier (en: keymap))
- `timedatectl` (date et heure du système, timezone, NTP)
- `systemd-analyze` (analyse de la séquence de démarrage, verification des fichiers de config de systemd)
- `coffeectl` (faire le café)

Quizz : quelles informations possède un processus ?

Quizz : quelles informations possède un processus ?

- son code (a.k.a. zone de texte)
- zone de données
- tas
- pile
- registres
- pointeurs de où on en est (instructions, pile)
- son état (sleeping, running, stopped, dead, zombie, ...) `man ps`
- son PID, le PID de son parent (PPID)
- son user propriétaire (UID) et son groupe propriétaire (GID)
  - (raffiné en EUID, SUID, RUID, EGID, SGID, RGID)
- variables d'environnement : PATH, ...
- arguments de sa commande
- répertoire courant
- les fichiers ouverts et leurs descripteur (un identifiant entier)
- sa niceness, limites de ressources qu'il peut utiliser
- des informations venues de l'extérieur
- ...

Consulter cours S&R, le répertoire `/proc/<pid>/` et `man 1 ps`

Quizz : quelles sont les façons d'echanger des informations avec/entre les processus ?

Quizz : quelles sont les façons d'echanger des informations avec/entre les processus ?

- tubes nommés (cf S&R)
- sockets (cf S&R) : sockets UNIX et sockets Berkley
- variables d'environnement (du parent vers l'enfant)
- arguments de la commande (du parent vers l'enfant)
- exit code (de l'enfant vers le parent)
- stdin/stdout/stderr (utilisation des tubes anonymes)
- signaux
- l'écriture/lecture dans un fichier commun maison

Redirections

Terminal et terminal virtuel (ne pas confondre avec le shell), fonctionnement ligne à ligne

`stdin`, `stdout`, `stderr`

redirections simples : `|` `<` `>` `>>` `2>` `2>>`

commandes utiles : `tee [-a]`, `xargs`, `sponge` (du paquet `moreutils`)

Démo à partir des commandes de manipulation de texte de la page du cours.

Signaux

signaux, `kill`, `man 7 signal`

exemple : `man 8 nginx`

Jobs : processus liés à un shell

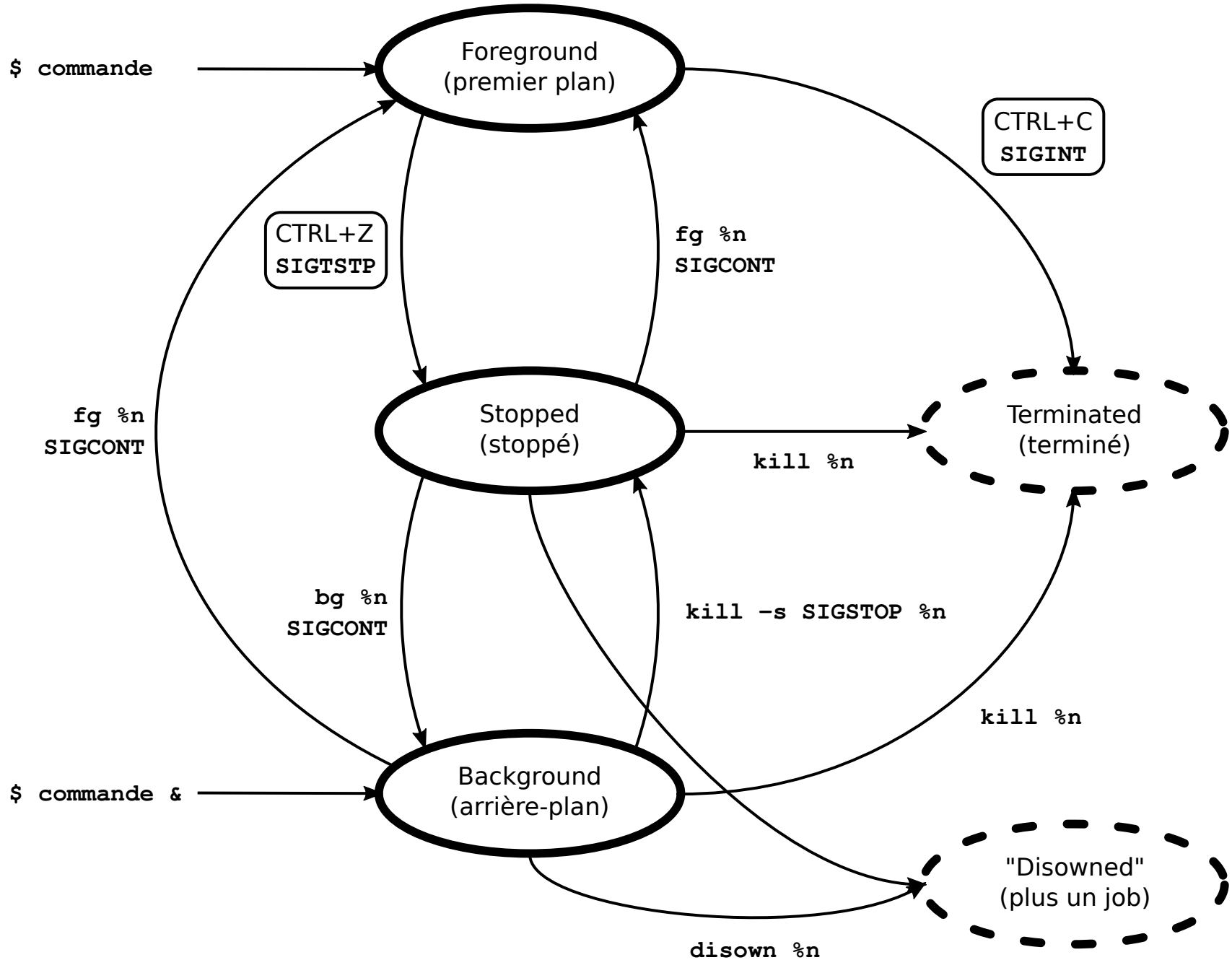
Le shell est un processus.

Le shell regroupe les processus d'un même pipeline sous forme de "jobs".

Tant qu'ils sont attachés au shell, il est possible de les terminer, les stopper, les faire passer au permier/arrière plan, ne pas leur faire suivre `SIGHUP` lors de la fermeture du terminal, à l'aide de racourcis clavier (qui correspondent à l'envoi de signaux).

Voir et tester l'automate sur la page suivante, par exemple en lançant un éditeur graphique `gedit` le stopper, le passer en tâche de fond, le disowner, fermer le terminal brutalement, etc, tout en faisant des `pstree` et des `ps -eo stat` pour voir son statut et sa filiation.

Jobs


$ commande
Foreground
(premier plan)
CTRL+C
SIGINT
CTRL+Z
SIGTSTP
fg %n
SIGCONT
fg %n
SIGCONT
Stopped
(stoppé)
Terminated
(terminé)
kill %n
bg %n
SIGCONT
kill -s SIGSTOP %n
kill %n
$ commande &
Background
(arrière-plan)
"Disowned"
(plus un job)
disown %n

Cahier de week-end

Quelques blagues (réelles ou supposées) à rajouter au cahier de vacances :

- https://turnoff.us/geek/signals/
- https://turnoff.us/geek/dont-sigkill/
- https://turnoff.us/geek/dont-sigkill-2/
- https://turnoff.us/geek/one-last-question
- https://turnoff.us/geek/forked/
- https://www.luc-damas.fr/hop/the-fork-is-strong-in-your-family