# Notion de processus

Cette page et les suivantes du chapitre sont insiprées des pages suivantes :

- https://www.lecluse.fr/nsi/NSI\_T/archi/process/
- https://www.lyceum.fr/tg/nsi/3-architectures-materielles-systemes-dexploitation-et-reseaux/2-gestion-des-processus-par-un-systeme-dexploitation
- http://lycee.educinfo.org/index.php?page=creation\_thread&activite=processus

et bien entendu de l'habituel Numérique et Sciences Informatique, 24 leçons avec exercices corrigés aux editions Ellipses.

L'objectif de cette partie est de comprendre comment une machine peut exécuter plusieurs tâches de manière simultanées : si l'on prend l'exemple du smartphone, alors que nous regardons une vidéo, il va suivre les antennes relais et se synchroniser avec, écouter s'il y a un appel téléphonique ou des SMS qui arrive, vérifier les nouveaux courriers électroniques, mettre à jour les notifications des différents réseaux sociaux, etc.

Pour comprendre, il nous faudra nous poser les questions suivantes :

- 1. Comment sont donc exécutés les programmes par le système d'exploitation?
- 2. Comment plusieurs programmes peuvent-ils être exécutés simultanément?
- 3. Quels sont les risques et problèmes soulevés par ces exécutions simultanées ?

# 1. Du programme au processus

# **E** Les processus

Un processus est un programme en cours d'exécution sur un ordinateur. Il est caractérisé par

- un **ensemble d'instructions** à exécuter souvent stockées dans un fichier sur lequel on clique pour lancer un programme (par exemple firefox.exe)
- un **espace mémoire** dédié à ce processus pour lui permettre de travailler sur des **données** qui lui sont propres : si vous lancez deux instances de Firefox, chacune travaillera indépendamment de l'autre avec ses propres données.
- des ressources matérielles : processeur, entrées-sorties (accès à internet en utilisant la connexion Wifi).

Il ne faut donc pas confondre le fichier contenant un **programme** (portent souvent l'extension .exe sous windows) et le ou les processus qu'ils engendrent quand ils sont exécutés : un programme est juste un fichier contenant une suite d'instructions (firefox.exe par exemple) alors que les processus sont des instances de ce programme ainsi que les ressources nécessaires à leur exécution (plusieurs fenêtres de Firefox ouvertes en même temps).

Il est possible de visualiser et gérer les processus actifs d'une machine par l'intermédiaire d'un **gestionnaire de processus**, qui est un programme spécifique au système d'exploitation :

- sur windows, en utilisant \_^ Ctrl \_+ \_ <- Alt \_+ \_ <- Del \_, mais pas au Lycée... La gestion des processus étant critique, seuls les administrateurs de la machine peuvent y accéder...
- sur linux simplement par l'utilisation d'un terminal (la ligne de commande)

# Manipulations

- 1. /bin/: commandes de base nécessaires au démarrage et à l'utilisation d'un système minimaliste.
  - /sbin/: Exécutables pour les administrateurs (abréviation de system binaries, soit binaires système en français).
  - /usr/bin/: Binaires exécutables qui ne sont pas déjà présents dans /bin et donc pas indispensables à un système minimaliste.

Sous linux, les programmes sont par convention situés dans les dossiers bin (pour binaries en anglais):

Utiliser la commande 1s pour lister les programmes présents dans /usr/bin/.

- 2. Utilisez la commande cat /bin/ls pour afficher le contenu du programme ls . On constate que le fichier est un fichier compilé (utilisez | ^ Ctrl |+| C |pour revenir au prompt).
- 3. Pour lancer un programme, il suffit d'écrire son **nom** (sans préciser le chemin si le dossier a été ajouté aux variables d'environnement). Par exemple vous pouvez lancer un interpréteur python par la commande python3 (tapez exit() pour sortir de l'interpréteur).
- 4. Vous pouvez visualiser les processus en exécution par l'intermédiaire de la commande ps :
  - a. ps simplement listera vos propres processus utilisateurs
  - b. ps -1 donnera plus de détails
  - c. ps -a -u -x ou ps -aux affichera tous les processus de tous les utilisateurs(-a), avec le propriétaire du processus (-u), ainsi que les processus qui ne sont pas attachés à un terminal (-x).
  - d. ps -ef affichera tous les processus (every) avec toutes les informations disponibles (full).
  - e. Le manuel d'utilisation de la commande ps est obtenu par la commande man ps (ou par wikipedia, comme d'habitude).

# 2. Création d'un processus

# **E** Création des processus

La création d'un processus peut intervenir

- au démarrage du système;
- par un appel d'un autre processus;
- par une action d'un utilisateur (lancement d'application).

 $Quand \ un \ processus \ est \ cr\'{e}e, \ le \ syst\`{e}me \ d'exploitation \ lui \ aloue \ un \ identifiant \ unique: \ le \ {\color{red} PID \over PID} \ (\textit{Process Identifier}).$ 

Sur Linux, la création d'un processus se fait par clonage d'un autre processus au travers d'un appel systeme fork().

- le processus qui fait appel à fork() est appelé processus père;
- le processus qui est ainsi créé par clonage est le processus fils ;
- après le clonage, un processus peut remplacer son programme par un autre programme grâce à l'appel système exec().

Lorsqu'un processus engendre un fils, l'OS génère un nouveau PID pour le fils, et le fils connait aussi le numéro de son père : le **PPID** (*Parent Process Identifier*)

# Lecture d'une commande ps -ef

Voici une capture d'écran du début d'une commande ps -ef :

```
fabien@fabien-ThinkPad-X270-W10DG:~/Documents/GitHub/ZoneNSI.md$ ps -efa
                     PPID C STIME TTY
                                                   TIME CMD
                                               00:00:02 /sbin/init splash
00:00:00 [kthreadd]
root
                         Θ
                           0 13:10 ?
                           0 13:10
root
                         0
                           0 13:10 ?
                                               00:00:00
root
                                                         [rcu_gp]
                                                         [rcu_par_gp]
[slub_flushwq]
                           0 13:10 ?
root
                                               00:00:00
root
                              13:10
                                               00:00:00
                                               00:00:00
                           0 13:10 ?
                                                         [netns]
root
                           0 13:10 ?
root
                                               00:00:00 [mm_percpu_wq]
                              13:10
                                               00:00:00
                                                         [rcu_tasks_rude
root
                                                         [rcu_tasks_trace]
               12
                           0 13:10
                                               00:00:00
root
                           0 13:10 ?
                                               00:00:00 [ksoftirqd/0]
root
root
               14
                              13:10
                                               00:00:08
                                                         [rcu_sched]
               15
                                               00:00:00
                                                         [migration/0]
root
                            0
                              13:10
```

Les différentes colonnes affichées sont :

- UID : l'identifiant du propriétaire du processus ;
- PID : tel que défini ci-dessus ;
- PPID : le PID du parent;
- STIME : l'heure de démarrage du processus ;
- TIME : le temps d'exécution du processus ;
- CMD : le programme exécuté par le processus.



### Le premier processus ?

Sous un système d'exploitation comme Linux, au moment du démarrage de l'ordinateur un tout premier processus (appelé processus 0 ou encore Swapper) est créé à partir de "rien" (il n'est le fils d'aucun processus). Ensuite, ce processus 0 crée un processus souvent appelé init ou systemd, il est créé au démarrage du système.

À partir de ce premier processus, les processus nécessaires au bon fonctionnement du système d'exploitation Linux sont créés (par exemple les processus crond, inetd, getty,...). Puis d'autres processus sont créés à partir des fils de init ...

Ce système de création un peu particulier (désigné souvent par fork/exec) conduit à l'émergence d'une arborescence de processus : un processus père engendre un ou plusieurs fils qui à leur tour engendrent des fils etc...

L'instruction pstree permet de visualiser l'arbre de processus :

```
-networkd-dispat
-packagekitd---2*[{packagekitd}]
-polkitd---2*[{polkitd}]
-rsyslogd---3*[{rsyslogd}]
-rtkit-daemon---2*[{rtkit-daemon}]
-systemd——(sd-pam)
                                                       -at-spi-bus-laun---dbus-daemon
-3*[{at-spi-bus-laun}]
-at-spi2-registr---2*[{at-spi2-registr}]
                                                       -dbus-daemon
                                                        -dconf-service---2*[{dconf-service}]
                                                      -evolution-addre—2 [{dcoin service];
-evolution-addre—5*[{evolution-addre}]
-evolution-calen—8*[{evolution-calen}
-evolution-sourc—3*[{evolution-sourc}]
-evolution-sourc—45*[/medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//medocs—5*//m
                                                                                                                                                     bash—mkdocs—5*[{m
-bash—pstree
-4*[{gnome-terminal-}]
                                                         -anome-terminal--
                                                                                                                                                                                                                                                 -5*[{mkdocs}]
                                                        -goa-daemon----3*[{goa-daemon}]
                                                       -goa-daemon—3*[{goa-daemon}]
-goa-identity-se -2*[{goa-identity-se}]
-gvfs-udisks2-vo—3*[{gvfs-udisks2-vo}]
-gvfsd—gvfsd-trash—2*[{gvfsd-trash}]
-2*[{gvfsd}]
-gvfsd-fuse—5*[{gvfsd-fuse}]
-gyfsd-metadata—2*[{gvfsd-metadata}]
                                                          -obexd
                                                       -pipewire——{pipewire}
-pipewire-media-——{pipewire-media-}
-pulseaudio——4*[{pulseaudio}]
                                                       -xdg-desktop-por—5*[{xdg-desktop-por}]
-xdg-desktop-por—3*[{xdg-desktop-por}]
                                                       -xdg-document-po—fusermount3

-6*[{xdg-document-po}]

-xdg-permission-—2*[{xdg-permission-}]
                                                  _xreaderd—_2*[{xreaderd}]
  -systemd-journal
  -sýstemd-ĺogind
   systemd-resolve
```

#### Chercher les PID

#### Enoncé

- 1. A l'aide de la commande ps -ef | more, chercher les PID et PPID des processus:
  - a. systemd
  - b. cron
- 2. Ouvrez un deuxième terminal Linux puis :
  - a. Dans le 1er, ouvrez l'interpréteur Python grâce à la commande python3
  - b. Dans le 2ème, cherchez le PID et le PPID des processus python3



Une des commandes les plus utiles sous Linux est la commande top. Cette commandes affiche les processus actifs **en temps réel**. L'application est plus riche qu'il n'y parait. Il faut passer un peu de temps à explorer toutes les options. Celles-ci s'activent par des raccourcis clavier. En voici quelques uns :

- H: affiche l'aide
- M : trie la liste par ordre décroissant d'occupation mémoire. Pratique pour repérer les processus trop gourmands
- P: trie la liste par ordre décroissant d'occupation processeur
- [I]: filtre les processus inactifs. Cela ne montre que ceux qui travaillent réellement.
- [K]: permet de tuer un processus à condition d'en être le propriétaire. Essayez de tuer systemd ...
- V : permet d'avoir la vue arborescente sur les processus.
- Q: permet de quitter top

# 3. États d'un processus

Un processus n'a pas seulement besoin d'accéder au processeur, mais il a souvent besoin d'accéder à des ressources autres comme :

- la mémoire vive : RAM,
- une mémoire de masse : disques durs, clés USB, mémoire flash...
- la lecture ou l'écriture d'un fichier...
- les périphériques d'entrée et de sortie : clavier, souris, écran, imprimante...

Ces ressources externes étant beaucoup moins rapides que le processeur, elles bloquent les processus lors de leur exécution.

Ainsi, lors de la vie d'un processus, celui-ci peut passer par trois états :

- PRET (ready): le processus est lancé et attend l'accès au processeur.
- ELU (running) : le processus a obtenu l'accès au processeur : il peut s'exécuter.
- BLOQUE (sleeping): le processus est en cours d'exécution, mais attend une ressource en mémoire par exemple, il quitte le processeur pour libérer les ressources.



### Exemple

Imaginons un simple programme Python:

```
texte = input("Entrez une phrase ?")
print(f"La longueur de votre phrase est {len(texte)}")
```

Lorsqu'on exécute ce programme, un processus est crée, et passe en état PRET. Dès qu'il a accès au processeur, il passe dans l'état Elu. La première instruction est alors exécutée. Or celle ci attend une saisie au clavier, et le processus ne peut pas continuer tant que l'utilisateur n'a pas agit. L'OS met alors en attente le processus (en état BLOQUE).

Lorsque l'OS, qui gère aussi les entrées et les sorties, repère une interaction avec le clavier, il va alors débloquer le processus, qui repassera en état PRET, puis en état ELU et exécutera alors la deuxième instruction. Sans autre intervention, le processus passera en état final (stopped), et se terminera naturellement. Il libèrera alors les ressources mémoire qu'il occupait, et enverra un signal à son processus parent, afin que celui-ci l'élimine de la table des processus.

# 4. Terminer un processus

### **E** Terminer un processus

Il est possible d'envoyer un signal de terminaison à un processus actif, par l'intermédiaire de la commande kill, en lui précisant le PID du processus qu'on souhaite arrêter. Un tel signal est équivalent au fait de fermer une fenêtre en interface graphique (par la croix ou bien par 2 Alt + F4), ou bien au raccourci ^ Ctrl + C | dans la console.

Un tel signal est intercepté par l'application du processus et géré par cette dernière. Par exemple un logiciel de traitement de texte peut alors proposer à l'utilisateur de sauvegarder ses fichiers avant de quitter.

# Question

- 1. Lancez l'interpréteur python3 dans un terminal.
- 2. Depuis un autre terminal, repérer le PID correspondant à cette instance de python3 (processus de type ipython)
- 3. Terminer le processus.

# **E** Tuer un processus

Parfois un processus se comporte de manière anormale, par exemple par une trop grande consommation de mémoire, ou lorsqu'une application « ne répond plus ». Il est possible dans ce cas de tuer le processus en question, en lui envoyant un signal de terminaison sans que l'application puisse l'intercepter, en passant l'option -9 à la commande kill, par exemple kill -9 13259. En reprenant l'exemple du traitement de texte, dans ce cas les modifications ne seront pas sauvegardées et des données risquent d'être perdues. C'est une commande à utiliser avec précaution.



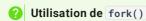
#### Création et suppression de processus en Python

Nous allons créer un processus à partir de Python. Pour cela :

1. Téléchargez dans un terminal le fichier long\_time.py, contenant les lignes suivantes :

```
import time
a = 0
for i in range(100000):
   a += a**3
   time.sleep(0.01)
print("terminé")
```

- 2. Ouvrez un deuxième terminal.
- 3. Dans le premier, utilisez la commande  ${\tt python3\ long\_time.py}$  .
- 4. Dans le second, lancez la commande ps, et cherchez le processus correspondant à l'exécution du script long\_time.py.
- 5. Tuez le processus avec la commande kill, et observez ce qui se passe dans les deux terminaux.
- 6. Recommencez la procédure à partir de 4 en tuant le processus avec la commande kill -9, et observez les deux terminaux.



1. Téléchargez le fichier testFork.py, dans votre dossier personnel sous Ubuntu, à l'aide de la commande wget suivie du lien du fichier.

Ce fichier contient les lignes suivantes :

```
# Python program to explain os.fork() method
# importing os module
import os, time
# Create a child process
# using os.fork() method
pid = os.fork()
# pid greater than 0 represents
# the parent process
if pid > 0 :
    print("I am parent process:")
    print("Process ID:", os.getpid())
   print("Child's process ID:", pid)
# pid equal to 0 represents
# the created child process
else :
   print("\nI am child process:")
    print("Process ID:", os.getpid())
    print("Parent's process ID:", os.getppid())
a = 0
for i in range(10000):
    a += a**3
   time.sleep(0.001)
print(f"Finished {os.getpid()}")
# If any error occurred while
# using os.fork() method
# OSError will be raised
```

- 2. Exécutez ce fichier par la commande python3 testFork.py. Observez dans ce terminal l'effet du script.
- 3. Dans un autre terminal, avec la commande ps -ef, observez les processus créés, ainsi que leurs PID et PPID.
- 4. (Si nécessaire, relancez le script!) Tuez le processus enfant. Que se passe-t-il?
- 5. Relancez le script! Tuez le processus parent et attendez quelques instants. Que se passe-t-il?

# Processus Zombies

Parfois un processus père est tué avant que ses processus fils soient terminés. Ceux-ci restent alors dans la table des processus en situation finale, mais ne sont pas supprimés. On parle alors de **processus zombies**. Ceux-ci occupent une partie de la mémoire, tout en étant devenus inutiles...