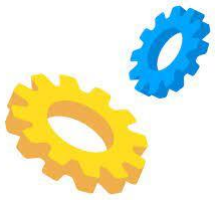


Chap 2. Gestion de processus et services

Un système d'exploitation se compose de processus. Ces derniers, responsables de la stabilité et la sécurité du système, sont exécutés dans un ordre bien précis et observent des liens de parenté entre eux. On distingue deux catégories de processus, ceux axés sur l'environnement utilisateur et ceux sur l'environnement matériel.

Les processus

A. Principes

Lorsqu'un programme s'exécute, le système va créer un processus qui lui est associé en plaçant les données et le code du programme en mémoire et en créant **une pile d'exécution**. De ce fait un processus est une instance (dynamique) d'un programme auquel est associé un environnement processeur (CO, PSW, registres) et un environnement mémoire.

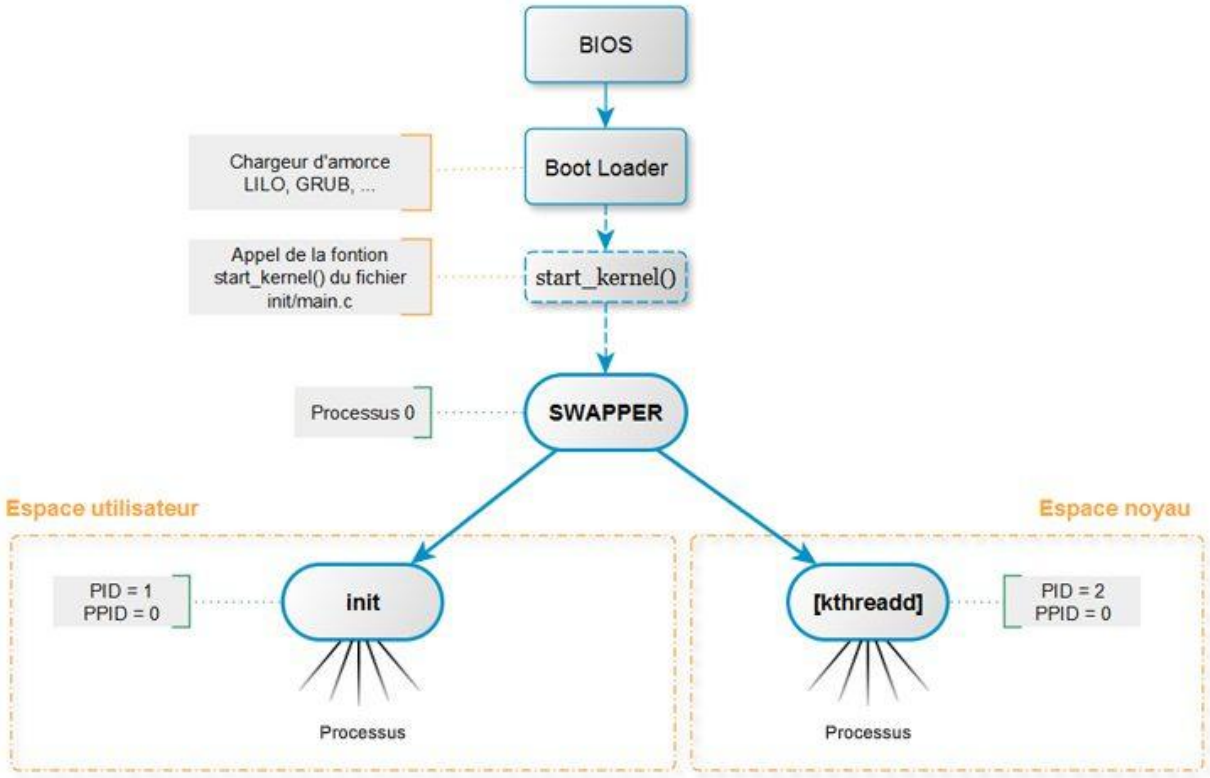
Au sein des systèmes UNIX, les processus sont identifiés par leur PID (ou Process Identifiant, un identifiant unique) et y sont organisés de façon hiérarchique. Il existe une relation père / fils entre les processus, un processus fils est le résultat de l'appel système de la primitive fork() par le processus père qui duplique son propre code pour créer un fils. Le PID du fils est renvoyé au processus père pour qu'il puisse dialoguer avec. Chaque fils possède l'identifiant de son père, le **PPID** (ou *Parent Process Identifiant*).

Note : Les processus ne sont pas à confondre avec les threads (dit aussi processus légers). Chaque processus (parent, enfant) possède son propre contexte mémoire (ressources et espace d'adressage) alors que les threads issus d'un même processus partagent ce même contexte.

B. Création des processus

Avant le lancement du système, le chargeur de démarrage (ou d'amorce) fait appel à la fonction **start_kernel()** du fichier **init/main.c**. Cette fonction va créer le tout premier processus : le **swapper** (ou **Processus 0**, ou encore **sched pour scheduler**) qui occupera la première entrée de la table des processus. Le swapper va ensuite créer deux autres processus, le processus init et le processus [kthreadd] et va s'endormir.

À partir de ce moment-là, nous pouvons considérer qu'il existe 2 espaces au sein du système. Un espace utilisateur avec **init** au sommet de la hiérarchie et un espace noyau avec **[kthreadd]**.



Notons que **Init** et **[kthreadd]** étant tout les deux des fils du swapper ont leur PPID égal à 0. La commande **ps** permet de le vérifier.

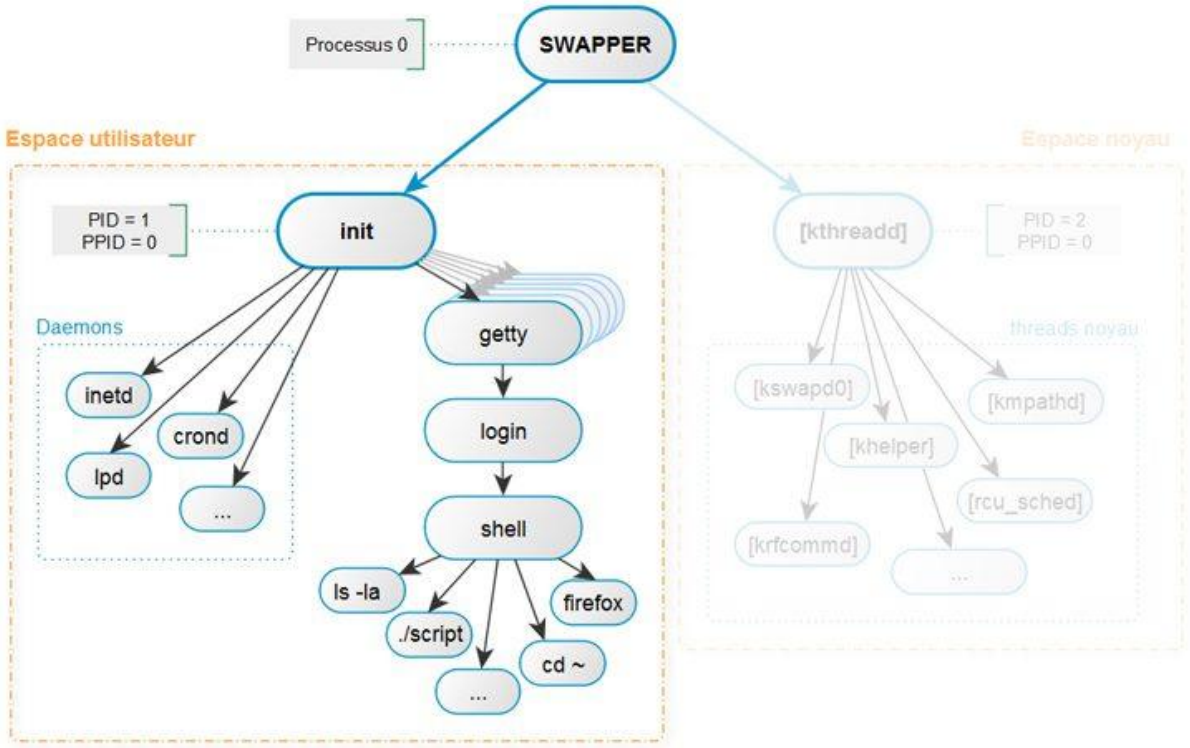
```
ps -aef

UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 12:01 ?        00:00:01 /sbin/init
root         2     0  0 12:01 ?        00:00:00 [kthreadd]
```

Arborescences

A. Espace utilisateur

Au sommet de l'espace utilisateur se trouve le processus **init**, ayant le **PID n°1**, qui est le premier à avoir été lancé. Il se chargera ensuite de créer les autres processus nécessaires au fonctionnement du système, notamment les processus daemons et les processus gettys qui à leur tour créeront d'autres processus fils et ainsi de suite. Init sera ici l'ancêtre de tous les processus.



1. Les processus gettys

Les processus getty (**get tele**type) sont chargés de la surveillance des terminaux. Ils permettent entre autres aux utilisateurs de se connecter.

```
ps -aef

UID        PID  PPID  C STIME TTY          TIME CMD
root       850     1  0 08:58 tty4     00:00:00 /sbin/getty -8 38400 tty4
root       854     1  0 08:58 tty5     00:00:00 /sbin/getty -8 38400 tty5
root       862     1  0 08:58 tty2     00:00:00 /sbin/getty -8 38400 tty2
```

Quand un utilisateur veut se connecter à un terminal, le processus getty va créer un nouveau processus fils, le processus login. Ce dernier va vérifier à l'aide du fichier **/etc/passwd** la correspondance login/mot de passe et les autorisations de l'utilisateur. Si la connexion est réussie, le processus login va à son tour créer un nouveau processus fils, le processus shell (interpréteur de commandes). Il va analyser si l'utilisateur entre une commande et créera un nouveau processus chargé de l'exécuter. Ce dernier processus prendra fin lorsque la commande (ou le programme lancé en ligne de commande) se terminera.

```
ps axjf

 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 4141 24877 24877 24877 ?           -1 Ss       0   0:00 \_ sshd: root@pts/0
24877 24882 24882 24882 pts/0    26592 Ss       0   0:00 |   \_ -bash
24882 26592 26592 24882 pts/0    26592 R+       0   0:00 |       \_ ps axjf
 4141 26275 26275 26275 ?           -1 Ss       0   0:00 \_ sshd: root@pts/2
26275 26280 26280 26280 pts/1    26984 Ss       0   0:00     \_ -bash
26280 26984 26984 26280 pts/1    26984 S+       0   0:00         \_ /bin/bash ./testDodo.sh
26984 27004 26984 26280 pts/1    26984 S+       0   0:00           \_ sleep 1
```

2. Les processus daemons

Les daemons sont des processus, pour la plupart lancés au démarrage du système, qui restent en tâche de fond jusqu'à ce qu'on fasse appel à eux. Leurs noms se terminent généralement par la lettre d (pour daemon).

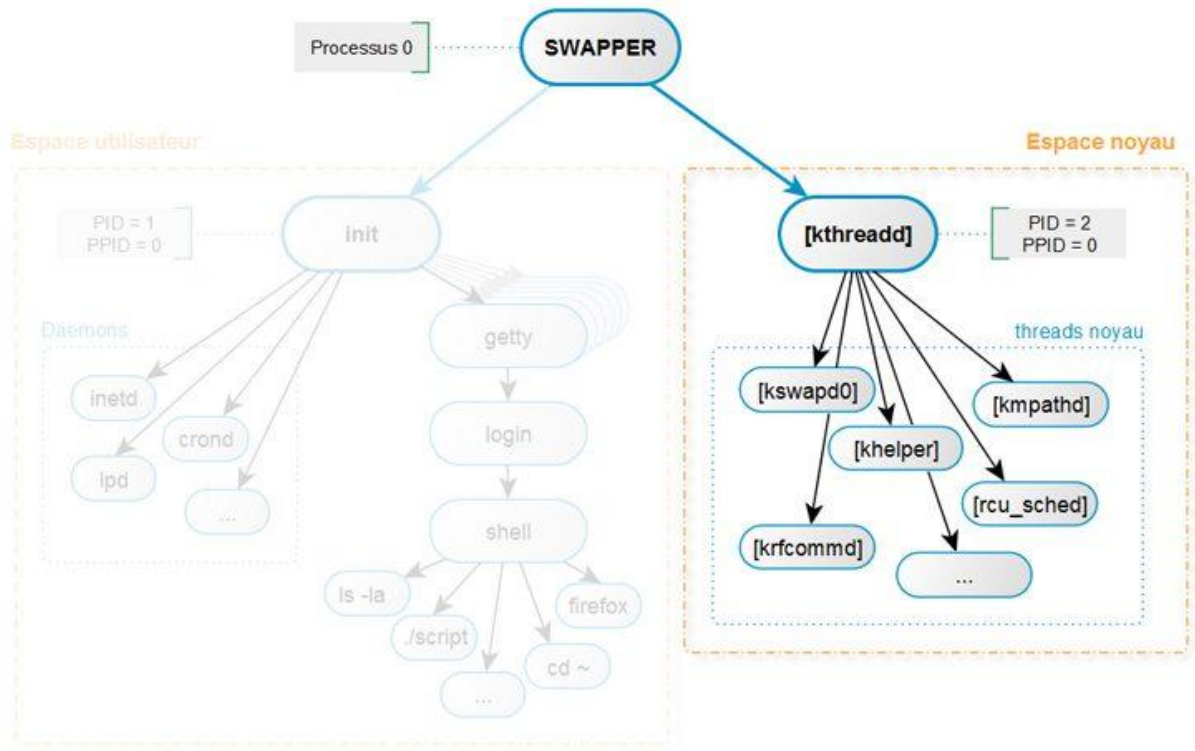
```
ps -aef

UID        PID  PPID  C STIME TTY          TIME CMD
root      3920     1  0 mai06 ?        00:00:41 /usr/sbin/rsyslogd
root      4086     1  0 mai06 ?        00:00:00 /usr/sbin/vsftpd
root      4141     1  0 mai06 ?        00:00:01 /usr/sbin/sshd
```

Parmi les plus connus, nous avons par exemple crond (tâches planifiées), inetd (surveillance réseau), rsyslogd (logs du système), lpd (gestion de l'imprimante), etc.

B. Espace Noyau

Cet espace est occupé par les **threads noyau** (ou **kthreads**, ou encore processus noyau) qui gèrent la partie hardware du serveur (c'est pourquoi ils sont lancés depuis le swapper) et ont une priorité haute. Ils s'exécutent dans l'espace d'adressage du noyau.



Au sommet de cet espace sur trouve le daemon **[kthreadd]**, processus qui se chargera de lancer tous les autres kthreads, reconnaissables de par leurs noms entre crochets [], **[kthreadd]** est donc l'unique père de cette hiérarchie et de ce fait tous les kthreads présents dans cet espace (sauf [kthreadd] lui même) ont un PPID de 2.

```
ps axjf

 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S        0   0:03 \_ [ksoftirqd/0]
    2     5     0     0 ?           -1 S<       0   0:00 \_ [kworker/0:0H]
    2     7     0     0 ?           -1 S        0   0:00 \_ [migration/0]
    2     8     0     0 ?           -1 S        0   0:00 \_ [rcu_bh]
    2     9     0     0 ?           -1 S        0   1:11 \_ [rcu_sched]
    2    10     0     0 ?           -1 S        0   0:00 \_ [migration/1]
    2    11     0     0 ?           -1 S        0   0:00 \_ [ksoftirqd/1]
[...]
```

Parmi les plus connus nous avons par exemple [kswapd0] (daemon du swap), [kupdate] (cache disque), etc.

IV. Conclusion

C'est le système qui, normalement, se charge de la gestion des processus, cependant il est possible d'intervenir avec l'aide de commandes comme **ps, top, htop**. Ces commandes s'avèrent

très pratiques en cas de problèmes (consommation mémoire excessive, processus zombie, …). Elles permettent dans la plupart des cas un contrôle assez étendu et d'avoir une bonne vision d'ensemble sur le système et les processus.

En résumé :

- Un **processus** est un programme en cours d'exécution.
- Chaque processus a un **PID** (Process ID), un **PPID** (Parent PID), et tourne avec un utilisateur.

Commandes de base

ps aux # Affiche tous les processus

top / htop # Vue dynamique des processus

pgrep <nom> # Trouve le PID d'un processus

pidof <commande> # Trouve le PID d'une commande

Suivi et tri des processus

- Utiliser **top, htop, ps** avec tri sur la mémoire, CPU, utilisateur...

Contrôle des processus

kill <PID> # Envoie un signal (SIGTERM par défaut)

kill -9 <PID> # SIGKILL (forcé)

killall <nom_processus> # Tue tous les processus d'un nom donné

nice -n <valeur> commande # Lance avec une priorité donnée

renice <valeur> -p <PID> # Modifie la priorité

Services sous Linux et systemd

Qu'est-ce qu'un service ?

- Un processus lancé en arrière-plan, souvent au démarrage, géré par systemd.

Fichiers unitaires (unit files)

- Localisation : /etc/systemd/system/, /lib/systemd/system/
- Structure :

[Unit]

Description=Mon service personnalisé

[Service]

ExecStart=/usr/bin/mon_script.sh

[Install]

WantedBy=multi-user.target

Commandes systemctl

systemctl start <service> # Démarrer un service

systemctl stop <service> # Arrêter un service

systemctl restart <service> # Redémarrer un service

systemctl status <service> # État du service

systemctl enable <service> # Activation au démarrage

systemctl disable <service> # Désactivation

systemctl list-units --type=service

Surveillance et logs

journalctl

journalctl -xe # Erreurs récentes

journalctl -u <service> # Logs d'un service

journalctl --since "1 hour ago" # Logs récents

Autres outils

- lsof, strace, systemd-analyze