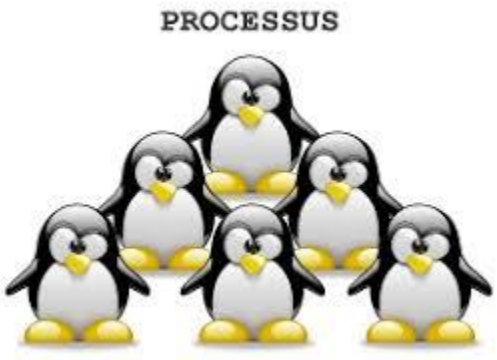

GESTION DES PROCESSUS

Serveur Linux CentOS

Table des matières

1 Caractéristiques d'un processus

Plusieurs processus peuvent s'exécuter en parallèle. Le système doit être capable de les identifier. Pour cela il attribue à chacun d'entre eux, un numéro appelé **PID** (Process ID / Identifiant du processus).

Un processus peut lui-même créer un autre processus, il devient donc un processus parent ou père, et le nouveau processus, un processus enfant. Ce dernier est identifié par son **PID**, et le processus père par son numéro de processus appelé **PPID** (Parent Processus Identification).

Tous les processus sont ainsi identifiés par leur **PID**, mais aussi par le **PPID** du processus qui la créé, car tous les processus ont été créés par un autre processus. Le seul qui ne suit pas cette règle est le premier processus lancé : le processus **init** qui n'a pas de père et qui a pour **PID** 1.

2 États des processus

Un processus devient tour à tour actif/inactif tout au long de sa vie ; il passe donc par différents états :

État	Description
Élu	Le CPU exécute le code de ce processus.
Prêt	Le processus dispose de toutes les ressources nécessaires à son fonctionnement à l'exception du CPU. Il est dans une file d'attente associée au CPU.
Bloqué	Le processus est en attente d'une ressource autre que le CPU ou d'un événement.
Zombie	Le processus a terminé son exécution et est prêt à mourir ; à ce stade, l'ordonnanceur a détecté l'arrêt de la tâche mais le père du processus n'a pas encore pris en compte sa mort ; il existe donc toujours une entrée pour le processus dans la table des processus.

3 Information sur les processus

3.1 Commande time

La commande **time** permet d'afficher le temps d'exécution d'un processus.

```
[root@localhost ~]# time gzip anaconda-ks.cfg

real    0m0.037s
user    0m0.000s
sys     0m0.002s
```

La signification des trois lignes retournées par la commande **time** est :

Champ	Signification
real	Temps réel écoulé lors de l'exécution
user	Temps d'utilisation CPU en mode utilisateur
sys	Temps d'utilisation CPU en mode noyau

3.2 Commande pidof

La commande **pidof** permet d'afficher le **process id** d'un processus.

```
[root@localhost ~]# pidof vi

12006
```

3.3 Commande pgrep

La commande **pgrep** permet rechercher les **process id** en se basant sur des critères de recherche spécifiques.

Rechercher les processus appartenant à un utilisateur spécifique (hakimb):

```
[root@localhost ~]# pgrep -u hakimb

12035
12074
```

Rechercher les processus appartenant à un utilisateur spécifique(hakimb) et ayant un nom spécifique(vi):

```
[root@localhost ~]# pgrep -u hakimb vi
12074
```

3.4 Commande pstree

La commande **pstree** permet d'afficher l'arborescence des processus (parent-enfant).

Cette commande n'est pas installée dans la version minimale. Pour l'installer :

```
[root@localhost ~]# yum install psmisc
```

Pour exécuter la commande **pstree** :

```
[root@localhost ~]# pstree

systemd─┬─NetworkManager─┬─dhclient
        │                └─2*[{NetworkManager}]
        ├─auditd───{auditd}
        ├─chronyd
        ├─crond
        ├─dbus-daemon
        ├─login───bash
        ├─lvmetad
        ├─master─┬─pickup
        │        └─qmgr
        ├─polkitd───6*[{polkitd}]
        ├─rsyslogd───2*[{rsyslogd}]
        ├─sshd───sshd───bash───pstree
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-udevd
        └─tuned───4*[{tuned}]
```

Pour afficher le PID :

```
[root@localhost ~]# pstree -p

systemd(1)─┬─NetworkManager(3658)─┬─dhclient(3709)
           │                      ├─{NetworkManager}(3683)
           │                      └─{NetworkManager}(3685)
           ├─auditd(3619)───{auditd}(3620)
           ├─chronyd(3671)
           ├─crond(3672)
           ├─dbus-daemon(3645)
           ├─login(3676)───bash(4422)
           ├─lvmetad(3478)
           ├─master(4129)─┬─pickup(4130)
           │              └─qmgr(4131)
           ├─polkitd(3641)─┬─{polkitd}(3650)
           │               ├─{polkitd}(3651)
           │               ├─{polkitd}(3652)
           │               ├─{polkitd}(3653)
           │               ├─{polkitd}(3654)
           │               └─{polkitd}(3655)
           ├─rsyslogd(3933)─┬─{rsyslogd}(4088)
           │                └─{rsyslogd}(4089)
           ├─sshd(3930)───sshd(4438)───bash(4440)───pstree(4507)
           ├─systemd-journal(3458)
           ├─systemd-logind(3644)
           ├─systemd-udevd(3487)
           └─tuned(3929)─┬─{tuned}(4402)
                         ├─{tuned}(4403)
                         ├─{tuned}(4404)
                         └─{tuned}(4418)
```

Pour afficher une branche de l'arborescence, il suffit de donner le PID :

```
[root@localhost ~]# pstree 3930

sshd───sshd───bash───pstree
```

3.5 Commande ps

La commande **ps** permet d'afficher les informations sur les processus en cours.

Pour afficher la liste des processus de la sessions courante :

```
[root@localhost ~]# ps

   PID TTY          TIME CMD
 11906 pts/0    00:00:00 bash
 12754 pts/0    00:00:00 ps
```

Pour afficher la liste des processus de toutes les sessions:

```
[root@localhost ~]# ps -e

   PID TTY          TIME CMD
     1 ?        00:00:01 systemd
     2 ?        00:00:00 kthreadd
     3 ?        00:00:00 ksoftirqd/0
     5 ?        00:00:00 kworker/0:0H
     7 ?        00:00:00 migration/0
     8 ?        00:00:00 rcu_bh
     9 ?        00:00:00 rcu_sched
    10 ?        00:00:00 lru-add-drain
```

Remarque

Vous pouvez aussi utiliser la commande interactive **top** pour afficher des informations détaillées sur les processus.

4 Arrêter un processus

Pour pouvoir arrêter un processus, vous devez connaître son **PID.**

Un utilisateur ne peut arrêter que ses propres processus. Seul l'administrateur système a le droit d'arrêter un processus ne lui appartenant pas.

4.1 Commande kill

La commande **kill** termine un ou plusieurs processus référencés par leurs **PID**.

La syntaxe est la suivante :

kill *options liste des pid*

Option	Description
-l	Affiche la liste des signaux.
-signal	Le *signal* (référencé par son numéro ou par son nom) est envoyé aux processus. Le signal 9 (KILL) tue systématiquement les processus.

Pour afficher la liste des signaux qui peuvent être envoyés à un processus :

```
[root@localhost ~]# kill -l
 1)SIGHUP       2)SIGINT       3)SIGQUIT
 4)SIGILL       5)SIGTRAP      6)SIGABRT
 7)SIGBUS       8)SIGFPE       9)SIGKILL
10)SIGUSR1     11)SIGSEGV     12)SIGUSR2
13)SIGPIPE     14)SIGALRM     15)SIGTERM
16)SIGSTKFLT   17)SIGCHLD     18)SIGCONT
19)SIGSTOP     20)SIGTSTP     21)SIGTTIN
22)SIGTTOU     23)SIGURG      24)SIGXCPU
25)SIGXFSZ     26)SIGVTALRM   27)SIGPROF
28)SIGWINCH    29)SIGIO       30)SIGPWR
31)SIGSYS      34)SIGRTMIN    35)SIGRTMIN+1
36)SIGRTMIN+2  37)SIGRTMIN+3  38)SIGRTMIN+4
39)SIGRTMIN+5  40)SIGRTMIN+6  41)SIGRTMIN+7
42)SIGRTMIN+8  43)SIGRTMIN+9  44)SIGRTMIN+10
45)SIGRTMIN+11 46)SIGRTMIN+12 47)SIGRTMIN+13
48)SIGRTMIN+14 49)SIGRTMIN+15 50)SIGRTMAX-14
51)SIGRTMAX-13 52)SIGRTMAX-12 53)SIGRTMAX-11
54)SIGRTMAX-10 55)SIGRTMAX-9  56)SIGRTMAX-8
57)SIGRTMAX-7  58)SIGRTMAX-6  59)SIGRTMAX-5
60)SIGRTMAX-4  61)SIGRTMAX-3  62)SIGRTMAX-2
63)SIGRTMAX-1  64)SIGRTMAX
```

EXEMPLE

Pour terminer le processus de vi, on doit trouver le pid de vi :

```
[root@localhost ~]# pidof vi

12006
```

Ensuite terminer le processus :

```
[root@localhost ~]# kill 12006
```

4.2 Commande pkill

La commande **pkill** permet de terminer tous les processus d'un utilisateur spécifique.

```
[root@localhost ~]# pkill -u hakimb
```

4.3 Commande killall

La commande **killall** permet de terminer tous les processus correspondant à un non spécifique.

```
[root@localhost ~]# killall vi
```

5 Changer la priorité d'un processus

Les processus tournent avec un certain degré de priorité, un processus plus prioritaire aura tendance à accaparer plus souvent les ressources du système pour arriver le plus vite possible au terme de son exécution. C'est le rôle du système d'exploitation de gérer ces priorités.

5.1 Commande nice

Vous disposez de la commande **nice** pour modifier la priorité d'un processus. La syntaxe est la suivante :

```
nice valeur commande
```

Plus le nombre est grand, plus la priorité est faible. Par exemple **-2** est plus prioritaire que **0**.

La fourchette de valeur dépend du système **UNIX**. Sur **RedHat**, cette fourchette se situe dans l'intervalle **-20** (le plus prioritaire) à **19** (le moins prioritaire).

```
[root@localhost ~]# nice -5 xclock
```

```
[root@localhost ~]# nice  -n  -20  xclock
```

On peut l'utiliser par exemple pour compiler un programme.

```
[root@localhost ~]# nice -5 cc monprogramme.c
```

Si aucun argument n'est fourni, **nice** affiche la priorité d'ordonnancement en cours sinon, **nice** exécute la commande désirée en ajustant la priorité d'ordonnancement.

Si aucun ajustement n'est précisé, la valeur de priorité de la commande est augmentée de **10**.

nice signifie **gentil**, la valeur de priorité considérée est une valeur de gentillesse. Plus celle-ci est élevée, plus le processus est gentille vis à vis des autres, leur laissant un accès plus fréquent à l'ordonnanceur.

5.2 Commande renice

La commande **renice** permet de changer la priorité d'un programme qui est déjà en exécution.

```
[root@localhost ~]# pidof xclock

13064
```

```
[root@localhost ~]# renice -8 13064

13064 (process ID) old priority 0, new priority -8
```

6 Exécuter un processus en tâche de fond

Pour lancer une commande quelconque, vous en entrez le nom après le prompt du Shell, tant que la commande n'est pas terminée, vous n'avez plus la main au niveau du Shell, vous ne disposez plus du prompt. Si la commande prend un certain temps, votre Shell ne vous donnera pas la main tant que la commande n'est pas terminée, vous êtes obligé de lancer un autre Shell, pour taper une autre commande.

Vous disposez d'une technique simple qui permet de lancer une commande à partir d'un Shell, et de reprendre aussitôt la main. Il vous suffit de rajouter un **&** à la fin de commande. Celle-ci se lancera en " tâche de fond ", et vous reviendrez directement au prompt du Shell.

6.1 Commande &

En tapant une commande en tâche de fond, vous aurez à l'affichage :

```
[root@localhost ~]# xclock  &
[1] 912
```

A la suite de la saisie de la commande suivie d'un **&**, le Shell vous donne immédiatement la main, et affiche le numéro du **PID** du processus lancé.

En lançant une commande à partir du Shell sans le **&** à la fin, et si celle-ci prend du temps à vous rendre la main, vous pouvez faire en sorte qu'elle bascule en tâche de fond, pour que vous repreniez la main.

```
[root@localhost ~]# xclock
```

Pour basculer en tâche de fond, il faut en premier suspendre le processus avec **CTRL+Z** ensuite le relancer en tâche de fond avec la commande **bg**.

```
[2]+  Stopped
```

6.2 Commande bg

La commande **bg** (background) fait de basculer le processus en arrière-plan:

```
[root@localhost ~]# bg
[2]+ top &
```

6.3 Commande fg

La commande **fg** (foreground) fait basculer processus en avant-plan:

```
[root@localhost ~]# fg
```

```
[root@localhost ~]# sleep 15000
```

```
[root@localhost ~]# sleep 15000  CTRL+Z
[1]+  Stopped                 sleep 15000
```

6.4 Commande jobs

La commande **jobs** permet d'afficher la liste des jobs qui s'exécutent en tâche de fond.

```
[root@localhost ~]# jobs
[1]+  Stopped                 sleep 15000
```

```
[root@localhost ~]# bg 1
[1]+ sleep 15000 &
```

```
[root@localhost ~]# jobs
[1]+  Running                 sleep 15000 &
```

```
[root@localhost ~]# fg
sleep 15000    CTRL+Z
[1]+  Stopped                 sleep 15000
```

```
[root@localhost ~]# bg
[1]+ sleep 15000 &
```

```
[root@localhost ~]# jobs
[1]+  Running                 sleep 15000 &
```

```
[root@localhost ~]# sleep 2000 &
[2] 6317
```

```
[root@localhost ~]# jobs
[1]-  Running                 sleep 15000 &
[2]+  Running                 sleep 2000 &
```

```
[root@localhost ~]# fg 2
sleep 2000     CTRL+Z
[2]+  Stopped                 sleep 2000
```

```
[root@localhost ~]# jobs
[1]-  Running                 sleep 15000 &
[2]+  Stopped                 sleep 2000
```

6.5 Commande nohup

Lancer un processus en tâche de fond sans se terminer si la session est fermée (**nohup**) :

EXEMPLE 1

```
[root@localhost ~]# nohup sleep 1000 &
[4] 6425

[root@localhost ~]# nohup: appending output to `nohup.out'
```

```
[root@localhost ~]# jobs
[1]   Running                 sleep 15000 &
[2]   Running                 sleep 2000 &
[3]-  Running                 nohup sleep 5000 &
[4]+  Running                 nohup sleep 1000 &
```

```
[root@localhost ~]# jobs
[1]   Running                 sleep 15000 &
[2]   Done                    sleep 2000
[3]-  Running                 nohup sleep 5000 &
[4]+  Done                    nohup sleep 1000
```

EXEMPLE 2

```
[root@localhost ~]# nohup ls &
[1] 6940

[root@localhost ~]# nohup: appending output to `nohup.out'

[1]+  Done                    nohup ls
```

```
[root@localhost ~]# more nohup.out
fichier1
fichier2
nohup.out
```