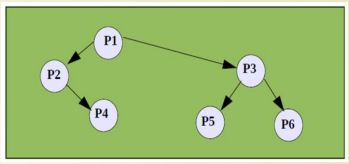


Linux
Les processus

JFA -160

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

02/10/2023

Introduction aux processus :

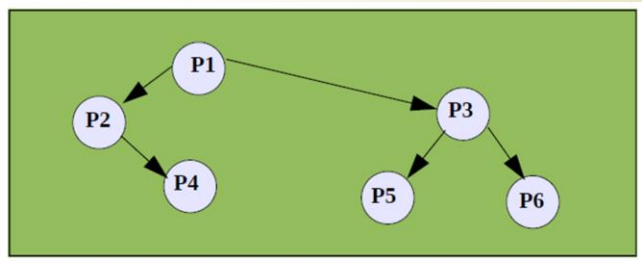
Un processus (process) est un programme (un ensemble d'octets en langage machine) en cours d'exécution dans un ordinateur.

Un système d'exploitation doit en général traiter plusieurs tâches en même temps. Comme il n'a, la plupart du temps qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme. Il traite une tâche à la fois, s'interrompt et passe à la suivante. La commutation de tâches étant très rapide, il donne l'illusion d'effectuer un traitement simultané.

JFA - 161

Les processus des utilisateurs sont lancés par un interprète de commande (Shell). Ils peuvent eux-mêmes lancer ensuite d'autres processus. On appelle le processus créateur, le père, et les processus créés, les fils. Les processus peuvent donc se structurer sous la forme d'une arborescence.

Au lancement du système, il n'existe qu'un seul processus, appelé processus «init» (P1), qui est l'ancêtre de tous les autres.



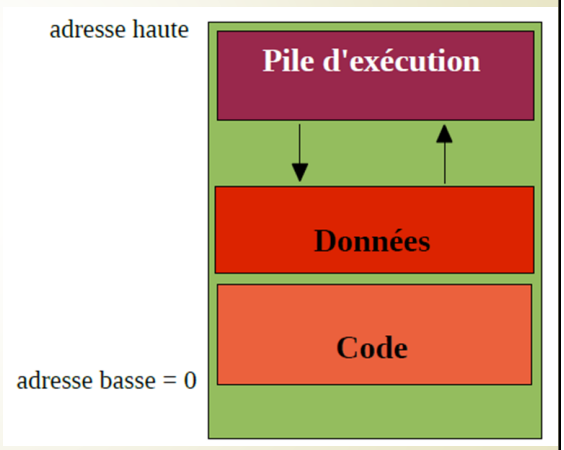
Introduction aux processus :

Les processus sont composés d'un espace de travail (espace d'adressage) en mémoire formé de 3 segments et visible par l'utilisateur/programmeur :

JFA - 162

- Le code correspond aux instructions, en langage d'assemblage, du programme à exécuter.
- La zone de données contient les variables globales ou statiques du programme ainsi que les allocations dynamiques de mémoire.
- Enfin, les appels de fonctions, avec leurs paramètres et leurs variables locales, viennent s'empiler sur la pile.

Les zones de pile et de données ont des frontières mobiles qui croissent en sens inverse lors de l'exécution du programme.



Le process Control Bloc :

Un processus est une entité active avec son propre compteur ordinal (*instruction pointer*) et l'ensemble des ressources qui lui sont associées. Ces informations sont stockées, pour chaque processus dans le PCB (*process control block*).

Le PCB va contenir typiquement :

JFA - 163

- L'ID du processus (PID), l'ID du processus parent (PPID) et l'ID de l'utilisateur du processus (UID) ;
- Les valeurs des registres correspondant au processus (l'état courant du processus, selon qu'il est élu, prêt ou bloqué) ;
- Le compteur ordinal du processus ;
- Le pointeur de pile : indique la position du prochain emplacement disponible dans la pile mémoire ;
- L'espace d'adressage du processus ;
- La liste des descripteurs de fichiers ;
- La liste de gestion des signaux ;
- D'autres informations telles que le temps CPU accumulé par le processus, etc.

Process Control Bloc

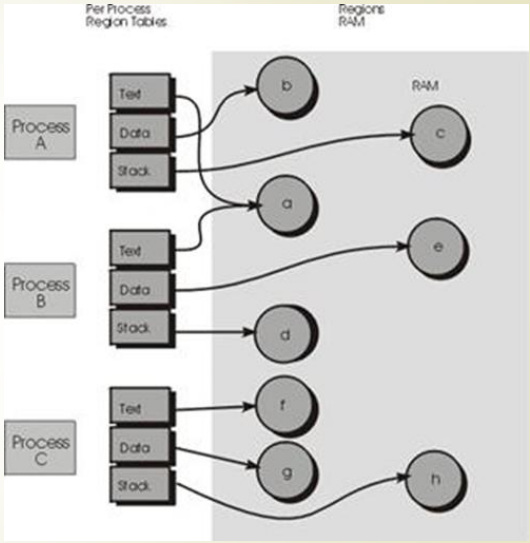
Processus Vs Programme :

Lors d'un changement de contexte, le processus en cours est arrêté et un autre processus peut utiliser le CPU. Le noyau doit arrêter l'exécution du processus en cours, copier les valeurs des registres hardware dans le PCB, et mettre à jour les registres avec les valeurs du nouveau processus. Et lancer l'exécution du nouveau processus.

JFA - 164

Un processus est un programme qui s'exécute et qui possède en plus, son compteur ordinal, ses registres et ses variables ; c'est là toute la subtilité entre programme et processus. La différence entre un processus et un programme est mince : le processus possède le programme mais également l'état courant de celui-ci dans la mémoire de l'ordinateur. Le programme est en fin de compte l'ensemble des fichiers qui, lorsqu'ils sont exécutés, deviennent le processus.

Un **processus** est une activité d'un certain type qui possède un **programme**, des données en entrée et en sortie, ainsi qu'un état courant.



02/10/2023

Format d'un fichier exécutable

Processus Vs Programme :

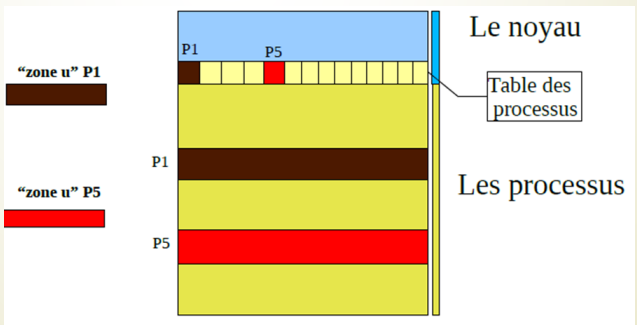
Un processus est donc un «programme» qui s'exécute et qui possède :

- Son propre compteur ordinal (@ de la prochaine instruction exécutable),
- Ses registres et ses variables.

Le concept de processus n'a donc de sens que dans le cadre d'un contexte d'exécution. Conceptuellement, chaque processus possède son propre processeur virtuel, en réalité, le vrai processeur commute entre plusieurs processus (**multiprogrammation**).

JFA - 165

Le noyau maintient une table, appelée « table des processus », pour gérer l'ensemble des processus (ici P1, ..., P5, ...). Cette table, interne au noyau, contient la liste de tous les processus avec des informations concernant chaque processus. C'est un tableau de structure « proc » (<sys/proc.h>).



02/10/2023

La zone U :

Le nombre des emplacements dans cette table des processus est limité pour chaque système et pour chaque utilisateur. Le noyau alloue pour chaque processus une structure appelée « zone u » (<sys/user.h>), qui contient des données privées du processus, uniquement manipulables par le noyau.

Seule la « zone u » du processus courant est manipulable par le noyau, les autres sont inaccessibles.

L'adresse de la « zone u » d'un processus est placée dans son mot d'état.

Le noyau dispose donc d'un tableau de structures (« proc.h ») dans la table des processus et d'un ensemble de structures (« user.h »), une par processus, pour piloter les processus.

JFA - 166

Le **contexte d'un processus** est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu :

son état (élu, prêt, bloqué, ...)
son mot d'état : en particulier
  * la valeur des registres actifs
  * le compteur ordinal
les valeurs des variables globales statiques ou dynamiques
son entrée dans la table des processus
sa « zone u »
les piles « user » et « system »
les zones de code (texte) et de données

02/10/2023

Le noyau et ses variables ne font partie du contexte d'aucun processus. L'exécution d'un processus se fait dans son contexte.

Changement de processus :

Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte (multiprogrammation).

Le noyau s'exécute alors dans le nouveau contexte.

Ce passage d'une tâche à une autre, est réalisée par un ordonnanceur (scheduler) au niveau le plus bas du système. Cet ordonnanceur est activé par des interruptions d'horloge, de disque et de terminaux.

A chaque interruption correspond un vecteur d'interruption, c'est-à-dire un emplacement mémoire contenant une adresse. L'arrivée de l'interruption provoque le branchement à cette adresse.

JFA - 167

Sur un intervalle de temps assez grand, tous les processus ont progressé, mais à un instant donné un seul processus est actif.

Comme le processeur commute entre les processus, la vitesse d'exécution d'un processus ne sera pas uniforme et variera vraisemblablement si les mêmes processus étaient exécutés à nouveau.

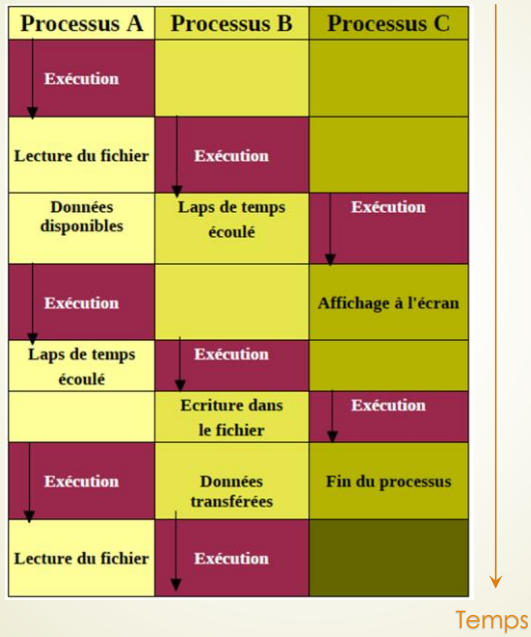
Il ne faut donc pas que les processus fassent une quelconque présomption sur le facteur temps.

Sur le schéma ci-dessous, les trois programmes deviennent trois processus indépendants qui ont chacun leur propre contrôle de flux (compteur ordinal).

02/10/2023

Processus :

JFA - 168



Processus :

Parmi les informations propres à chaque processus, qui sont contenues dans les structures système (« proc.h » et « user.h ») , on trouve :

JFA - 169

- un numéro d'identification unique appelé PID (Process Identifier), ainsi que celui de son père appelé PPID
- le numéro d'identification de l'utilisateur qui a lancé ce processus, appelé UID (User IDentifier), et le numéro du groupe auquel appartient cet utilisateur, appelé GID (Group IDentifier) ;
- le répertoire courant ;
- les fichiers ouverts par ce processus ;
- le masque de création de fichier, appelé umask ;
- la taille maximale des fichiers que ce processus peut créer, appelée ulimit ;
- la priorité ;
- les temps d'exécution ;
- le terminal de contrôle, c'est-à-dire le terminal à partir duquel la commande a été lancée.

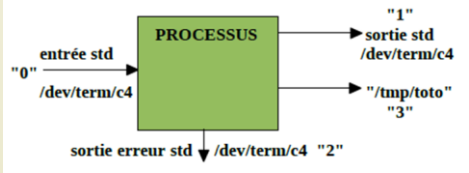
Processus :

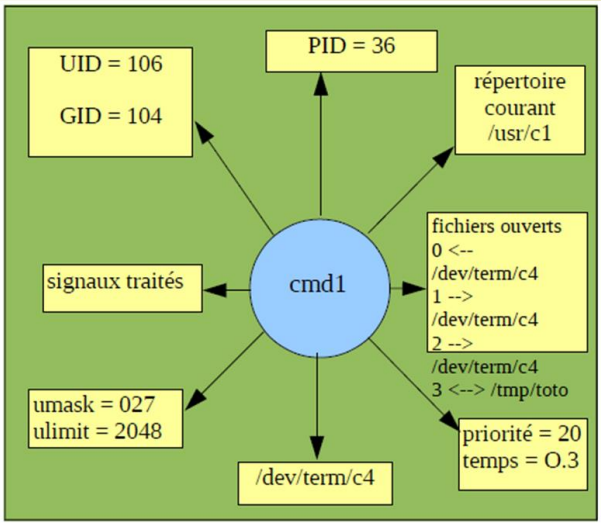
Voici un premier schéma (simplifié) d'un processus :

JFA - 170

Ce processus a le numéro *36*. Il a été lancé par l'utilisateur qui a *106* pour UID. Il est en train d'exécuter le programme '*cmd1*'. Il a *consommé 0.3* seconde, avec une priorité de *20*. Son masque de création de fichier est *027*. Son terminal de contrôle est */dev/term/c4*. Son répertoire courant est */usr/c1*.

Il a 4 fichiers ouverts :





02/10/2023

Métaphore :

Une métaphore illustrant la différence entre processus et programme :

Soit un informaticien qui prépare un gâteau d'anniversaire pour sa fille.

JFA - 171

- Il a une recette pour faire le gâteau et dispose de farine, d'oeufs, de sucre ...
- Ici la recette représente le programme (algorithme traduit en une suite d'instructions), l'informaticien joue le rôle du processeur (CPU) et les ingrédients sont les données à fournir.
- Le processus est l'activité de notre cordon bleu qui lit la recette, trouve les ingrédients nécessaires et fait cuire le gâteau.
- Si le fils de l'informaticien arrive en pleurant parce qu'il a été piqué par une guêpe, son père marque l'endroit où il était dans la recette (l'état du processus en cours est sauvegardé), cherche un livre sur les premiers soins et commence à soigner son fils.
- Le processeur passe donc d'un processus (la cuisine) à un autre plus prioritaire (les soins médicaux), chacun d'eux ayant un programme propre (la recette et le livre des soins).
- Lorsque la piqûre de la guêpe aura été soignée, l'informaticien reprendra sa recette à l'endroit où il l'avait abandonnée.

02/10/2023

Encore quelques commandes :

Certaines des caractéristiques de l'environnement peuvent être consultées par diverses commandes. Nous connaissons déjà :

- **pwd** affiche le chemin du répertoire courant
- **tty** affiche le terminal de contrôle
- **umask** affiche le masque de création de fichier

Voici d'autres commandes :

JFA - 172

❑ ***La commande id***

id consulte l'UID et le GID.

❖ ***Exemple :***

```
prompt> id
uid=106(c1) gid=104(cours)
prompt>
```

❑ ***La commande whoami***

whoami affiche uniquement le nom associé à l'UID.

❖ ***Exemple :***

```
prompt> whoami
 c1
prompt>
```


Encore quelques commandes :

❑ ***La commande pid***

Le **PID** est stocké dans une pseudo-variable spéciale que l'on appelle « **$** ».

On peut consulter le PID du Shell courant en tapant :

❖ ***Exemple :***

```
prompt> echo $$
36
prompt>
```

JFA - 173

Le 1er "**$**" définit le contenu de la pseudo- variable, alors que le second "**$**" correspond au PID du Shell courant.

Encore quelques commandes :

❑ ***La commande find***

find parcourt récursivement l'arborescence en sélectionnant des fichiers selon des critères de recherche, et exécute des actions sur chaque fichier sélectionné.

➢ ***Syntaxe générale***

```
find répertoire_de_départ critère_de_recherche action_à_exécuter
```

❖ ***Exemple :***

```
$ find $HOME -print
```

JFA - 174

Cette commande va parcourir toute l'arborescence à partir du répertoire de login (**$HOME**), va sélectionner tous les fichiers puisqu'il n'y a aucun critère de recherche, et va afficher le nom de chaque fichier trouvé.

➢ ***Le critère de recherche name***

-name `modèle sélectionne uniquement les fichiers dont le nom correspond au modèle.`

Attention ! Le modèle doit être interprété par la commande **find** et non par le shell, donc s'il contient des caractères spéciaux pour le shell (par exemple *), ceux-ci doivent être protégés.

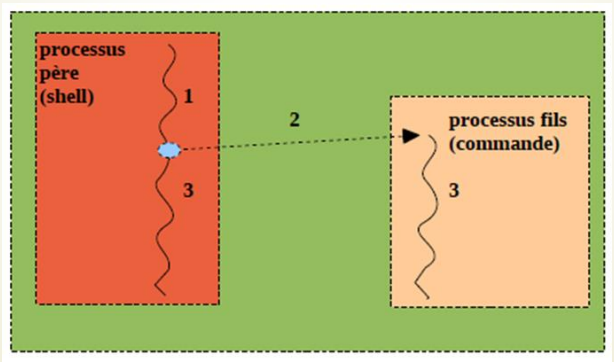
Création de processus :

Pour chaque commande lancée (sauf les commandes internes), le Shell crée automatiquement un nouveau processus.

Il y a donc 2 processus. Le premier, appelé processus père, exécute le programme Shell, et le deuxième, appelé processus fils, exécute la commande.

Le fils hérite de tout l'environnement du père, sauf bien sûr du PID, du PPID et des temps d'exécution.

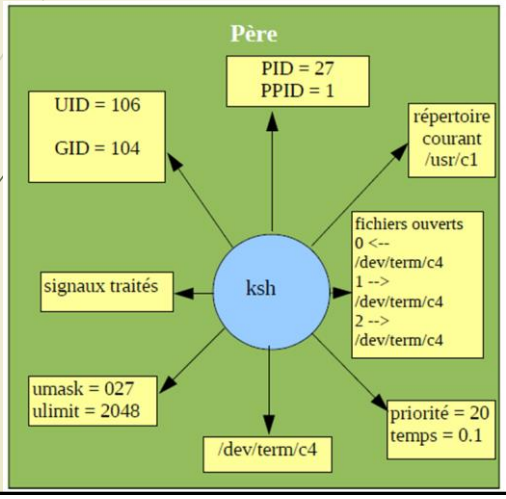
JFA - 175

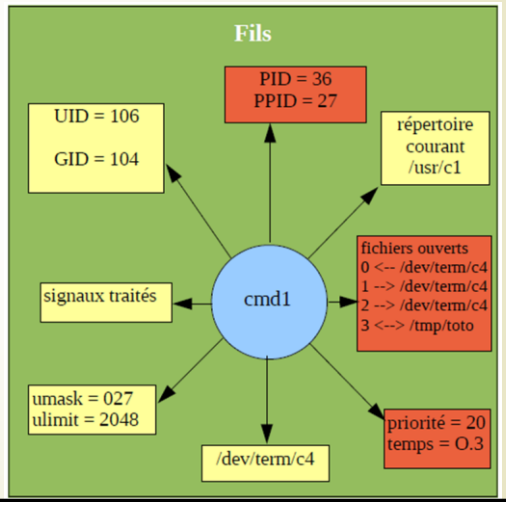


Création de processus :

JFA - 176

Un nouvel élément de l'environnement apparaît ici, le PPID. C'est le PID du processus père. Le père du processus 36 est le processus 27, et celui de 27 est le processus 1. Seul le fils (36) a ouvert le fichier /tmp/toto. Les deux processus peuvent parfaitement tourner en parallèle. La puissance de traitement est partagée entre tous les programmes lancés et, mis à part les machines multi- processeurs, un seul processus est actif à un instant t.



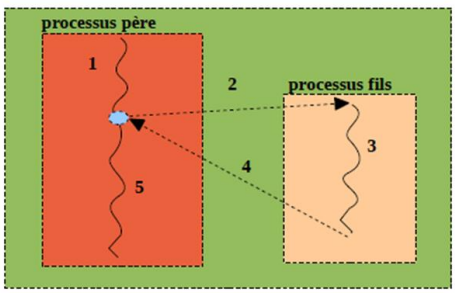


Enchainement de processus :

JFA - 177

On utilisera cette solution (processus lancés en parallèle) par exemple pour lancer un traitement très long, et continuer à travailler en même temps. Dans ce cas, on dit que le père a lancé un fils en tâche de fond (background) ou encore en mode asynchrone.

Une autre solution consiste à placer le processus père en attente jusqu'à ce que le processus fils soit terminé.



Pour lancer une commande en plaçant le père en attente, il suffit de taper la commande :

prompt> cmd1

... résultat de la commande cmd1

prompt>

Ce mode est donc le mode par défaut dans le Shell.

Tâche de fond :

Pour lancer une commande en tâche de fond, il faut faire suivre cette commande par le caractère '**&**':

```
prompt> cmd1 &
 [1] 127
prompt>
```

Le Shell affiche un numéro de tâche entre « [ ] » et le PID de cette tâche de fond, puis continue à travailler, donc affiche la chaîne d'invite et attend la prochaine commande.

En Bourne Shell, il n'y a pas de numéro de tâche, la même commande aurait donnée :

JFA - 178

```
prompt> cmd1 &
127
prompt>
```

Si vous avez plusieurs commandes successives à lancer en arrière plan, il faudra utiliser les parenthèses.

```
prompt> (cmd1; cmd2) &
 [2] 128
prompt>
```

La commande **cmd2** ne sera lancée que lorsque la commande **cmd1** sera terminée. Ceci dit, l'utilisateur récupère la main tout de suite. Le Shell détecte la présence du '**&**' partout sur la ligne.

Tâche de fond :

Dans le cas suivant, la commande cmd1 sera lancée par un sous-processus shell :

```
prompt> (cmd1)
prompt>
```

Dans le cas suivant, la commande **cmd1** est lancée en arrière plan et la commande **cmd2** est tout de suite lancée derrière, en direct (en parallèle).

```
prompt> cmd1 & cmd2
 [3] 130
prompt>
```

JFA - 179

La commande « wait n » permet d'attendre la mort de la tâche de fond dont le PID est « n ». .

```
prompt> cmd1 &
 [4] 132
prompt> wait 132
... on est bloqué jusqu'à ce que cmd1 se termine
prompt>
```

Si « n » n'est pas précisée, wait attend la mort de toutes les tâches de fond. wait ne s'applique qu'aux processus lancés dans le shell lui-même..

Arborescence de processus :

Tous les processus sont créés à partir d'un processus père, existant déjà.

Le premier processus est un peu spécial. Il est créé lorsque le système est initialisé. Il s'appelle "**init**", a le **PID 1** et n'est associé à aucun terminal. Son travail consiste à créer de nouveaux processus.

Le processus "**init**" crée 2 sortes de processus :

- des **démons**, c'est-à-dire des processus qui ne sont rattachés à aucun terminal, qui sont **endormis** la plupart du temps, mais qui se **réveillent** de temps en temps pour effectuer une tâche précise (par exemple la gestion des imprimantes).
- **des processus interactifs**, associés aux lignes d'entrées/sorties sur lesquelles sont rattachés des terminaux. Autrement dit des processus vous permettant de vous connecter.

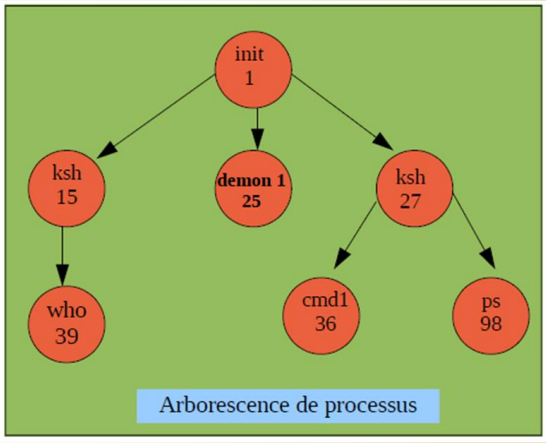
JFA - 180

02/10/2023

Arborescence de processus :

Pour visualiser les processus que vous avez lancé, tapez la commande «**ps**» :

```
prompt> echo $$
527
prompt> cmd1 &
prompt>
prompt> ps
PID TTY TIME COMMAND
527 ttyp4 1:70 -ksh
536 ttyp4 0:30 cmd1
559 ttyp4 0:00 ps
prompt>
```



JFA - 181

- PID identifie le processus,
- TTY est le numéro du terminal associé,
- TIME est le temps cumulé d'exécution du processus,
- COMMAND est le nom du fichier correspondant au programme exécuté par le processus.

02/10/2023

La commande ps :

Sans option, la commande concerne les processus associés au terminal depuis lequel elle est lancée.

- ps -ef # liste de tous les processus
- ps -ef | grep firefox # Firefox est-il actif ?
- ps aux # afficher les ressources utilisées
- ps -u root # les processus associés à un UID

JFA - 182

Il existe bien d'autres commandes pour gérer les processus,

comme par exemple la commande « top ».

La commande top :

Cette commande affiche en temps réel les processus qui consomment le plus de ressources systèmes. Dans les premières lignes, elle affiche des informations globales sur le système (charge, mémoire, nombre de processus, ...).

JFA - 183

```
top - 11:14:18 up  1:02,  1 user,  load average: 0,05, 0,09, 0,08
Tasks: 209 total,   1 running, 208 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,1 us,  0,2 sy,  0,0 ni, 99,8 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   7933,1 total,   6381,5 free,    797,0 used,    754,6 buff/cache
MiB Swap:   2048,0 total,   2048,0 free,      0,0 used.   6866,3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    607 systemd+  20   0   16004   6300   5472 S   0,3   0,1   0:03.81 systemd+
   1327 jfa       20   0 4053620 269836 129020 S   0,3   3,3   0:37.19 gnome-s+
   2104 jfa       20   0  227344   2432   2072 S   0,3   0,0   0:06.34 VBoxCli+
   2207 jfa       20   0  563848  54248  41684 S   0,3   0,7   0:02.81 gnome-t+
   2718 jfa       20   0 2819336  64620  49212 S   0,3   0,8   0:00.68 gjs
      1 root      20   0  167916  12340   8576 S   0,0   0,2   0:01.75 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par+
      5 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 netns
      6 root      20   0       0      0      0 I   0,0   0,0   0:01.01 kworker+
      7 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker+
      9 root       0 -20       0      0      0 I   0,0   0,0   0:00.06 kworker+
     10 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 mm_perc+
     11 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
     12 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
     13 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tas+
```

https://manpages.ubuntu.com/manpages/xenial/fr/man1/top.1.html

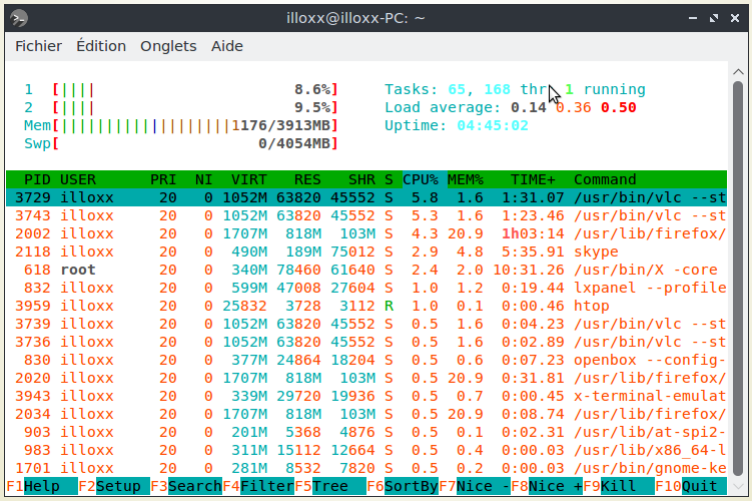
La commande htop :

HTOP est un outil logiciel à utiliser en ligne de commande.

```
sudo apt install htop
```

Il permet de superviser les ressources sur un serveur.

JFA - 184



https://doc.ubuntu-fr.org/htop

Retour de processus :

Lorsqu'un processus se termine, il retourne toujours une valeur significative ou statut.

Par convention, lorsqu'un processus se termine correctement, il retourne la valeur **0**, sinon il retourne une valeur différente de **0** (généralement **1**). Ce choix permet de ramener des codes significatifs pour différencier les erreurs.

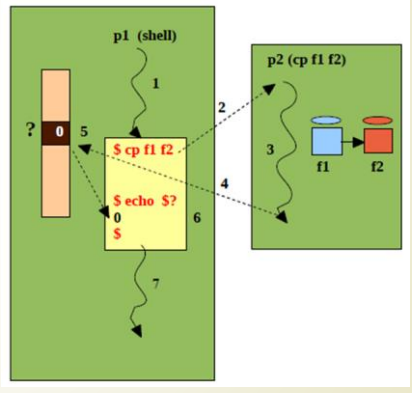
Le statut d'une commande Shell est placé dans la pseudo-variable spéciale, nommée « **?** ». On peut consulter sa valeur en tapant la commande :

```
echo $?
```

JFA - 185

Exemple :

```
prompt> cp f1 f2
prompt> echo $?
0
Prompt>
```



Sur cet exemple, la valeur 0 renvoyée par la commande « **echo** » nous indique que la commande « **cp** » s'est bien passée.

Autres commandes de gestion de processus :

❑ ***La commande kill***

kill envoie un signal à un (des) processus .

➢ ***Syntaxe générale***

```
kill [option] [PID]
```

La commande **kill** envoie un signal à des processus ou groupes de processus spécifiés, les obligeant à agir en fonction du signal. Lorsque le signal n'est pas spécifié, il est défini par défaut sur -15 (-TERM). The most commonly used signals are :

JFA - 186

- 1 (HUP) - Reload a process.
- 9 (KILL) - Kill a process.
- 15 (TERM) - Gracefully stop a process.

Pour obtenir une liste de tous les signaux disponibles, appelez la commande avec l'option -l

Exemple :

```
$ ps
    PID TTY          TIME CMD
   2226 pts/0    00:00:00 bash
   3614 pts/0    00:00:00 vi
   3617 pts/0    00:00:00 ps
$ kill -15 3614
```

```
$ ps
    PID TTY          TIME CMD
   2226 pts/0    00:00:00 bash
   3635 pts/0    00:00:00 ps
[1]+  Killed  vi toto
$
```

02/10/2023

Autres commandes de gestion de processus :

❑ ***La commande killall***

killall idem à kill mais pour tous les processus qui exécutent une commande spécifique .

➢ ***Syntaxe générale***

```
killall [option] Processus
```

La commande **killall** envoie un signal à tous les processus ou groupes de processus dont le nom est spécifié, les obligeant à agir en fonction du signal. Lorsque le signal n'est pas spécifié, il est défini par défaut sur -15 (-TERM).
The most commonly used signals are :

JFA - 187

- 1 (HUP) - Reload a process.
- 9 (KILL) - Kill a process.
- 15 (TERM) - Gracefully stop a process.

Pour obtenir une liste de tous les signaux disponibles, appelez la commande avec l'option -l

02/10/2023

Autres commandes de gestion de processus :

❑ ***La commande jobs***

jobs est une commande des systèmes d'exploitation Unix et Unix-like pour lister les processus lancés ou suspendus en arrière-plan.

➢ ***Syntaxe générale***

```
jobs [option] [jobID]
```

La commande **jobs** liste es processus en cours d'exécution ainsi que leur état : running ou stopped ou done,

JFA - 188

Exemple :

```
$ nano f1 &
$ firefox &
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$
```


Autres commandes de gestion de processus :

❑ ***La commande fg***

fg est la commande qui permet de remettre un processus au premier plan (foreground)

➢ ***Syntaxe générale***

```
fg [options] %[jobID]
```

Le **jobID** est le numéro fournit par la commande **jobs**

Exemple :

JFA - 189

```
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$ fg %2
Affiche la fenêtre Firefox !
```

Autres commandes de gestion de processus :

❑ *La commande bg*

bg est la commande qui permet de remettre un processus au arrière plan (background)

➢ ***Syntaxe générale***

```
bg [options] %[jobID]
```

Le **jobID** est le numéro fournit par la commande **jobs**

Exemple :

JFA - 190

```
$ jobs
[1]-  Stopped                 nano f1
[2]+  Running                 firefox &
$ bg %2
Remet la fenêtre Firefox en arrière plan !
```