

## Labo 3

Objectifs : Les commandes permettant la gestion de processus.

### Introduction

Maintenant que nous savons comment générer des processus via les *appels systèmes*, nous allons apprendre à les gérer au moyen des commandes du *shell*.

Afin de bien comprendre ce que chaque commande permet de faire, il est important que vous testiez chacune de celles-ci en reproduisant les exemples proposés et en testant d'autres options. Pour plus d'informations, référez-vous au manuel (<https://www.kernel.org/doc/man-pages/>).

### Lister les processus

ps [-][lujsvmaxSccewhrnu] [txx] [O[+ -]k1[+ -]k2...] [pids]		
Signification	Afficher l'état des processus en cours	
Description	Présente un cliché instantané des processus en cours. Pour obtenir un affichage remis à jour régulièrement, utilisez top.	
Options	-a	Affiche aussi les processus des autres utilisateurs
	-X	Affiche les processus de tous les terminaux
	-l	Affiche la liste longue
	-u	Affiche le nom de l'utilisateur et l'heure de démarrage
	-f	Affiche l'arbre des processus parents et enfants
	...	

#### EXEMPLES

Après avoir lancé le programme correspondant à la solution proposée dans le labo 2, et si on fait [CTRL+Z] afin de stopper l'exécution des 3 processus lancés, on peut afficher la liste des processus en cours au moyen de la commande ps. Le résultat de ces opérations est visible sur la figure 1.

```
piroc@svr24:~$ ./ex
PID 7019 - Debut du Pere
PID 7019 - Le PID du Fils 1 est 7020
PID 7019 - Le PID du Fils 2 est 7021
PID 7019 - Le Pere attend son Fils 1 [PID 7020]
PID 7021 - Debut du second fils [PPID 7019]
PID 7021 - Message 1 du second fils --
PID 7020 - Debut du premier fils [PPID 7019]
PID 7020 - Message 1 du premier fils -
PID 7020 - Message 2 du premier fils -
PID 7021 - Message 2 du second fils --
PID 7020 - Message 3 du premier fils -
PID 7020 - Message 4 du premier fils -
^Z
[1]+  Stoppé                  ./ex
piroc@svr24:~$ ps
  PID TTY          TIME CMD
 6906 pts/31    00:00:00 bash
 7019 pts/31    00:00:00 ex
 7020 pts/31    00:00:00 ex
 7021 pts/31    00:00:00 ex
 7068 pts/31    00:00:00 ps
piroc@svr24:~$
```

Figure 1 - Exemple d'utilisation de ps

Les figures 2 et 3 montrent deux utilisations de la commande ps.

```
piroc@svr24:~$ ps -l
F S    UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S    1002    6906  6905   0  80   0  -   1625 wait   pts/31    00:00:00 bash
0 T    1002    7019  6906   0  80   0  -    505 signal  pts/31    00:00:00 ex
1 T    1002    7020  7019   0  80   0  -    505 signal  pts/31    00:00:00 ex
1 T    1002    7021  7019   0  80   0  -    505 signal  pts/31    00:00:00 ex
0 R    1002    7291   6906   0  80   0  -   1134 -       pts/31    00:00:00 ps
```

Figure 2 - Exemple d'utilisation de ps -l

```
piroc@svr24:~$ ps -ux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
piroc     3147  0.0  0.7  10988  3828 ?        S    16:07   0:00 sshd: piroc@pts
piroc     3148  0.0  0.9   6488  4636 pts/6    Ss   16:07   0:00 -bash
piroc     3268  0.0  1.2   9300  6400 pts/6    S+   16:11   0:00 vi hello.c
piroc     6905  0.0  0.7  10988  3828 ?        S    17:04   0:00 sshd: piroc@pts
piroc     6906  0.0  0.9   6500  4648 pts/31    Ss   17:04   0:00 -bash
piroc     7019  0.0  0.1   2020   548 pts/31    T    17:05   0:00 ./ex
piroc     7020  0.0  0.0   2020    56 pts/31    T    17:05   0:00 ./ex
piroc     7021  0.0  0.0   2020    56 pts/31    T    17:05   0:00 ./ex
piroc     7379  0.0  0.4   4772  2460 pts/31    R+   17:07   0:00 ps -ux
```

Figure 3 - Exemple d'utilisation de ps -ux

Selon les options, la commande ps donne certaines des informations suivantes :

USER	indique le nom de l'utilisateur du processus
PID	numéro du processus
%CPU	affiche l'utilisation du processeur en pourcentage
%MEM	affiche l'utilisation de la mémoire vive en pourcentage
VSZ	donne l'utilisation des bibliothèques partagées et la mémoire utilisé pour son fonctionnement
RSS	donne l'utilisation de la mémoire physique utilisée en kilooctets par le processus (hors swap)
TTY	terminal contrôlant le processus
STAT	statut du processus <ul style="list-style-type: none"> <li>• <b>R</b> (runnable) s'exécutant ou pouvant s'exécuter</li> <li>• <b>S</b> (sleeping) en sommeil interruptible</li> <li>• <b>D</b> en sommeil non interruptible</li> <li>• <b>T</b> (traced) arrêté ou suivi</li> <li>• <b>X</b> tué</li> <li>• <b>Z</b> (zombie)</li> </ul>
START	indique l'heure à laquelle le processus a commencé
TIME	temps processeur utilisé par ce processus
COMMAND	commande avec tous ses arguments sous forme de chaîne.

#### REMARQUE

Pensez à utiliser grep pour limiter la liste.

Exemple : `ps -aux | grep ex` ne vous retournera que les processus contenant "ex".

Une autre manière d'afficher les processus en cours est de le faire en respectant leur hiérarchie...

<code>pstree [-a] [-c] [-h -Hpid] [-l] [-n] [-p] [-u] [-Z] [-A -G -U] [pid utilisateur]</code>	
Signification	Afficher l'arbre des processus. La racine est <code>init</code>
Description	Affiche les processus en cours d'exécution sous la forme d'un arbre. La racine de l'arbre est soit <code>pid</code> , soit <code>init</code> si <code>pid</code> est omis. Si un nom d'utilisateur est fourni, tous les arbres de processus ayant un processus racine appartenant à cet utilisateur seront affichés. Fusionne visuellement les branches identiques en les mettant entre crochets et en les préfixant par le nombre de répétition.
Options	<code>-a</code> Désactiver le compactage des branches
	<code>-c</code> Désactiver le compactage des sous-arbres identiques. Par défaut, les sous-arbres sont compactés chaque fois que c'est possible
	<code>-l</code> Afficher des lignes longues
	<code>-n</code> Trier par <code>pid</code> les processus ayant le même ancêtre plutôt que par nom
	<code>-p</code> Afficher les <code>pid</code>
	<code>-u</code> Afficher les transitions d'UID
	<code>-A</code> Utiliser les caractères ASCII pour dessiner l'arbre
	<code>-U</code> Utiliser les caractères UTF-8 pour dessiner l'arbre
	<code>-G</code> Utiliser les caractères VT100 pour dessiner l'arbre
	...

Un exemple de l'appel de cette commande est visible à la figure 4.

```
piroc@svr24:~$ pstree
systemd--acpid
          |--agetty
          |--atd
          |--cron
          |--dbus-daemon
          |--exim4
          |--ntpd
          |--rpc.idmapd
          |--rpc.statd
          |--rpcbind
          |--rsyslogd--{in:imklog}
                  |--{in:imuxsock}
                  |--{rs:main Q:Reg}
          |--sshd--sshd--sshd--bash
                  |--sshd--sshd--bash--pstree
          |--systemd-journal
          |--systemd-logind
          |--systemd-udev
```

Figure 4 – Exemple d'utilisation de `pstree`

## Tuer les processus

Avant de savoir quelle commande utiliser pour tuer un processus, il est important de comprendre la notion de flèche. Chaque processus sous UNIX est susceptible de réagir à des signaux qui lui sont envoyés.

*Une flèche est une forme limitée de communication entre processus utilisée par les systèmes de type Unix et ceux respectant les standards POSIX. Il s'agit d'une notification asynchrone envoyée à un processus pour lui signaler l'apparition d'un événement. Quand une flèche est envoyée à un processus, le système d'exploitation interrompt l'exécution normale de celui-ci. Si le processus possède une routine de traitement pour la flèche reçue, il lance son exécution. Dans le cas contraire, il exécute la routine de traitement des flèches par défaut.<sup>1</sup>*

Il existe 64 flèches différentes identifiées soit par leur numéro (en partant de 1), soit par leur nom symbolique. Les 32 flèches de rang le plus élevé sont des flèches temps réel, et ne seront pas utilisées dans le cadre de ces labos.

Cette liste est disponible au moyen de la commande suivante :

```
$ man 7 signal
```

Les premières flèches de cette liste sont présentées à la figure 5.

Voici tout d'abord les signaux décrits dans le standard POSIX.1-1990 original :

Signal	Valeur	Action	Commentaire
SIGHUP	1	Term	Déconnexion détectée sur le terminal de contrôle ou mort du processus de contrôle.
SIGINT	2	Term	Interruption depuis le clavier.
SIGQUIT	3	Core	Demande « Quitter » depuis le clavier.
SIGILL	4	Core	Instruction illégale.
SIGABRT	6	Core	Signal d'arrêt depuis abort(3).
SIGFPE	8	Core	Erreur mathématique virgule flottante.
SIGKILL	9	Term	Signal « KILL ».
SIGSEGV	11	Core	Référence mémoire invalide.
SIGPIPE	13	Term	Écriture dans un tube sans lecteur.
SIGALRM	14	Term	Temporisation alarm(2) écoulée.
SIGTERM	15	Term	Signal de fin.
SIGUSR1	30,10,16	Term	Signal utilisateur 1.
SIGUSR2	31,12,17	Term	Signal utilisateur 2.
SIGCHLD	20,17,18	Ign	Fils arrêté ou terminé.
SIGCONT	19,18,25	Cont	Continuer si arrêté.
SIGSTOP	17,19,23	Stop	Arrêt du processus.
SIGTSTP	18,20,24	Stop	Stop invoqué depuis le terminal.
SIGTTIN	21,21,26	Stop	Lecture sur le terminal en arrière-plan.
SIGTTOU	22,22,27	Stop	Écriture dans le terminal en arrière-plan.

Les signaux SIGKILL et SIGSTOP ne peuvent ni capturés ni ignorés.

Figure 5 - Quelques signaux...

Une flèche peut être généré par :

- le noyau (exemple : un processus accède une zone mémoire non autorisée - flèche SIGSEGV).
- l'appel système ou la commande kill.
- le processus lui-même (exemple : un processus initialise une alarme et lorsque que cette alarme expire, le noyau lui envoie la flèche SIGALRM).
- la frappe d'une touche par l'utilisateur du terminal (exemple : [CTRL-C] génère la flèche SIGINT dont l'action par défaut est de terminer le processus).

<sup>1</sup> [https://fr.wikipedia.org/wiki/Signal\\_%28informatique%29](https://fr.wikipedia.org/wiki/Signal_%28informatique%29)

## kill

La première commande permettant de tuer un processus est kill.

kill [ -s <i>signal</i> ] [ -l ] [ -- ] <i>pid</i> ...		
<b>Signification</b>	Envoyer une flèche au(x) processus dont on précise le(s) pid	
<b>Description</b>	Envoie la flèche indiquée aux processus mentionnés. Si on ne précise pas de flèche, SIGTERM est envoyé. Les flèches peuvent être indiquées soit par leur nom (par exemple -HUP ou -SIGHUP), soit par leur numéro (par exemple -1).	
<b>Options</b>	<i>pid</i>	Indique la liste des <i>pid</i> auxquels kill doit envoyer une flèche. Chaque <i>pid</i> peut prendre l'une des cinq formes suivantes :
	<i>n</i>	avec <i>n</i> supérieur à 0. Le processus de <i>pid n</i> recevra la flèche
	0	tous les processus du groupe de processus de l'appelant seront visés
	-1	tous les processus avec un <i>pid</i> supérieur à 1 recevront la flèche
	- <i>n</i>	où <i>n</i> est supérieur à 1. Tous les processus du groupe <i>n</i> sont visés
	<i>commande</i>	tous les processus invoqués en utilisant cette <i>commande</i> recevront la flèche
	-s	Indiquer la flèche à envoyer. Celui-ci peut être spécifié par son nom ou par son numéro.
	-l	Afficher une liste des noms de flèches connues. Ceux-ci sont fournis dans <code>/usr/include/linux/signal.h</code>

Une des options de kill liste les flèches connues. Le résultat de son appel est présenté à la figure 6.

```
piroc@svr24:~$ 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
```

Figure 6 – Signaux de `/usr/include/linux/signal.h`

### Raccourcis claviers liés aux processus

Ctrl-C	Arrête le processus en cours, celui qui a été lancé par la dernière commande.
Ctrl-Z	Stoppe le processus en cours, celui qui a été lancé par la dernière commande, mais ne le détruit pas : il reste en attente.

#### EXEMPLES

Dans un premier temps, exécutez le programme correspondant à la solution proposée dans le labo 2, faites [CTRL+Z] afin de stopper l'exécution du processus père, et affichez les processus en cours, comme montré ci-avant.

Utilisez la commande `kill` sur le processus père avec la flèche `-SIGKILL/-KILL/-9` pour le tuer sans attendre qu'il ait fini ce qu'il était en train de faire. Le résultat de ces opérations est visible sur la figure 7.

```
piroc@svr24:~$ ./ex
PID 26235 - Debut du Pere
PID 26235 - Le PID du Fils 1 est 26236
PID 26235 - Le PID du Fils 2 est 26237
PID 26235 - Le Pere attend son Fils 1 [PID 26236]
PID 26237 - Debut du second fils [PPID 26235]
PID 26237 - Message 1 du second fils --
PID 26236 - Debut du premier fils [PPID 26235]
PID 26236 - Message 1 du premier fils -
PID 26236 - Message 2 du premier fils -
^Z
[1]+  Stoppé                  ./ex
piroc@svr24:~$ ps
  PID TTY          TIME CMD
  9066 pts/14    00:00:00 bash
 26235 pts/14    00:00:00 ex
 26236 pts/14    00:00:00 ex
 26237 pts/14    00:00:00 ex
 26238 pts/14    00:00:00 ps
piroc@svr24:~$ kill -SIGKILL 26235
[1]+  Processus arrêté        ./ex
piroc@svr24:~$ ps
  PID TTY          TIME CMD
  9066 pts/14    00:00:00 bash
 26245 pts/14    00:00:00 ps
piroc@svr24:~$
```

Figure 7 - Exemple d'utilisation de `kill`

Si on avait lancé une flèche `-SIGHUP`, le père ne se serait pas arrêté tant que ses deux fils n'auraient pas fini leurs affichages...

## killall

Une autre commande permettant de tuer un processus est killall.

killall [-Z <i>patern</i> ] [-e] [-g] [-i] [-q] [-r] [-s <i>signal</i> ] [-u <i>user</i> ] [-v] [-w] [-l] [-V] [--] <i>nom</i> ...		
<b>Signification</b>	Envoyer une flèche à un processus dont on précise le nom	
<b>Description</b>	<p>Envoie une flèche à tous les processus en train d'exécuter les commandes mentionnées. Si aucune flèche n'est précisée, TERM ou SIGTERM est envoyé. Les flèches peuvent être indiquées soit par leur nom (par exemple -HUP ou -SIGHUP), soit par leur numéro (par exemple -1).</p> <p>Renvoie zéro si au moins un processus a reçu une flèche pour chaque commande mentionnée, ou aucune commande n'est mentionnée et au moins un processus correspond aux critères de recherche -u et -Z. killall renvoie un code non-nul dans tous les autres cas.</p> <p>Un processus killall ne s'envoie jamais de flèche à lui-même (mais il peut en envoyer à d'autres processus killall).</p>	
<b>Options</b>	-g	Tuer le groupe auquel le processus appartient. La flèche n'est envoyé qu'une seule fois au groupe, même si plusieurs processus lui appartenant ont été trouvés
	-i	Demander confirmation interactivement avant l'émission de la flèche
	-l	Afficher la liste de toutes les flèches connues
	-q	Ne pas se plaindre si aucun processus n'est tué
	-s	Envoyer la flèche fourni à la place de la flèche SIGTERM
	-u	Envoyer la flèche aux seuls processus appartenant à l'utilisateur spécifié. Les noms de commande sont optionnels
	-v	Afficher un compte-rendu de l'émission de la flèche
	...	

Une des options de la commande killall permet de lister les flèches connues. Le résultat de son appel est présenté à la figure 7.

```
piroc@svr24:~$ killall -l
HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM
STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH IO PWR SYS
UNUSED
```

Figure 8 - Exemple d'utilisation de killall -l

### EXEMPLES

En reprenant à nouveau l'exercice précédent, mais en lançant deux fois de suite le programme, on a généré deux pères qui ont chacun deux fils... On désire, après avoir stoppé ces processus au moyen de [CTRL+Z], tuer l'ensemble de ces processus.

Plusieurs solutions s'offrent à nous, mais la plus rapide en terme de commande est de déterminer le nom de ces processus, car naissant du même exécutable, ils ont le même nom de processus, et de tuer ces processus au moyen de la commande killall. Cette démarche est détaillée à la figure 9.

```

piroc@svr24:~$ ./ex
PID 1596 - Debut du Pere
PID 1596 - Le PID du Fils 1 est 1597
PID 1596 - Le PID du Fils 2 est 1598
PID 1596 - Le Pere attend son Fils 1 [PID 1597]
PID 1598 - Debut du second fils [PPID 1596]
PID 1598 - Message 1 du second fils --
PID 1597 - Debut du premier fils [PPID 1596]
PID 1597 - Message 1 du premier fils -
^Z
[1]+  Stoppé                  ./ex
piroc@svr24:~$ ./ex
PID 1599 - Debut du Pere
PID 1599 - Le PID du Fils 1 est 1600
PID 1599 - Le PID du Fils 2 est 1601
PID 1599 - Le Pere attend son Fils 1 [PID 1600]
PID 1601 - Debut du second fils [PPID 1599]
PID 1601 - Message 1 du second fils --
PID 1600 - Debut du premier fils [PPID 1599]
PID 1600 - Message 1 du premier fils -
PID 1600 - Message 2 du premier fils -
^Z
[2]+  Stoppé                  ./ex
piroc@svr24:~$ ps
  PID TTY          TIME CMD
 1564 pts/2    00:00:00 bash
 1596 pts/2    00:00:00 ex
 1597 pts/2    00:00:00 ex
 1598 pts/2    00:00:00 ex
 1599 pts/2    00:00:00 ex
 1600 pts/2    00:00:00 ex
 1601 pts/2    00:00:00 ex
 1602 pts/2    00:00:00 ps
piroc@svr24:~$ killall -KILL ex
[1]-  Processus arrêté        ./ex
[2]+  Processus arrêté        ./ex
piroc@svr24:~$ ps
  PID TTY          TIME CMD
 1564 pts/2    00:00:00 bash
 1604 pts/2    00:00:00 ps

```

Figure 9 - Exemple d'utilisation de killall



## Le "gestionnaire des tâches" sous Linux : top

La commande top permet d'avoir de l'information sur l'utilisation de la mémoire et des ressources du système en temps réel.

top [-] [d delay] [p pid] [q] [c] [C] [S] [s] [i] [n iter] [b]	
<b>Signification</b>	Afficher les processus
<b>Description</b>	<p>Les informations renvoyées par cette commande sont organisées de la façon suivante :</p> <ul style="list-style-type: none"> <li>Des informations sur le temps : l'heure actuelle, le temps depuis laquelle la machine tourne (<i>uptime</i>), le nombre d'utilisateurs ayant ouvert une session, la charge moyenne de la machine.</li> <li>Des informations sur les processus : nombre de processus au total, nombre de processus actifs, nombre de processus endormis, nombre de processus zombies, nombre de processus arrêtés.</li> <li>Des informations sur le processeur : le pourcentage d'utilisation suivant certaines catégories...</li> <li>Des informations sur la mémoire physique et virtuelle : <ul style="list-style-type: none"> <li>MEM : Mémoire physique (totale, utilisée, libre et tampons)</li> <li>SWAP : Mémoire virtuelle (totale, utilisée, libre et en cache)</li> </ul> </li> <li>Des informations sur chaque processus (voir ci-dessous)</li> </ul>
<b>Options</b>	d      Spécifier le délai entre deux rafraichissements
	p      Afficher les informations pour les <i>pid</i> listés
	q      Rafraichir l'affichage.
	c      Afficher la ligne de commande
	H      Afficher tous les threads.
	...
<b>Options interactives</b>	Lorsque le programme est en train de tourner, il est possible d'obtenir des informations supplémentaires grâce à certaines options dites "interactives" :
	q      Quitter
	M      Trier l'information par usage de la mémoire
	P      Trier l'information par usage du CPU
	N      Trier par numéro de processus
	m      Afficher/effacer l'information sur la mémoire
	k      Tuer un processus
	i      Ignorer les processus en attente ou zombie
	t      Afficher / effacer l'information sur l'usage du processeur
	l      Afficher/ effacer l'information <i>uptime</i>

Un exemple d'utilisation de cette commande est illustré à la figure 10.

```
top - 18:40:03 up 17 days,  3:33,  4 users,  load average: 0,08, 0,04, 0,01
Tasks:  92 total,   1 running,  91 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,3 us,   0,0 sy,   0,0 ni, 99,7 id,   0,0 wa,   0,0 hi,   0,0 si,   0,0 st
KiB Mem:  510296 total,  230484 used,  279812 free,   62884 buffers
KiB Swap: 1928188 total,   4156 used, 1924032 free. 100272 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1563	piroc	20	0	11120	4508	3812	S	0,3	0,9	0:04.76	sshd
1687	root	20	0	0	0	0	S	0,3	0,0	0:00.59	kworker/0:2
1	root	20	0	5384	3332	2656	S	0,0	0,7	0:29.40	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.21	kthreadd
3	root	20	0	0	0	0	S	0,0	0,0	1:10.78	ksoftirqd/0
5	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0,0	0,0	0:34.80	rcu_sched
8	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	migration/0
10	root	rt	0	0	0	0	S	0,0	0,0	0:17.60	watchdog/0
11	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	khelper
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kdevtmpfs
13	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	netns
14	root	20	0	0	0	0	S	0,0	0,0	0:00.49	khungtaskd
15	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	writeback
16	root	25	5	0	0	0	S	0,0	0,0	0:00.00	ksmd
17	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	crypto
18	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kintegrityd
19	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	bioaset
20	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kblockd
22	root	20	0	0	0	0	S	0,0	0,0	0:00.30	kswapd0
23	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	vmstat
24	root	20	0	0	0	0	S	0,0	0,0	0:00.00	fsnotify_mark
30	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kthrotld
31	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	ipv6_addrconf
33	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	deferwq
34	root	20	0	0	0	0	S	0,0	0,0	0:04.93	kworker/u64:1

Figure 10 - Exemple d'utilisation de la commande top

Les informations sur chaque processus sont constituées des colonnes suivantes :

- *pid* : le pid, le numéro identifiant de la tâche
- *USER* : l'utilisateur qui exécute ce processus
- *PR* : la priorité du processus
- *NI* : le *nice* du processus
- *VIRT* : la taille du code avec ses données et ses registres
- *RES* : taille de la mémoire physique
- *SHR* : taille de la mémoire partagée
- *S* : état du processus
- *%CPU* : utilisation du processeur en pourcentage
- *%MEM* : utilisation de la mémoire en pourcentage
- *TIME+* : temps d'utilisation
- *COMMAND* : commande ou programme qui est en cours d'exécution

## Les processus en arrière-plan

Lorsqu'une commande est lancée à partir d'un terminal, il faut normalement attendre que la commande soit terminée pour pouvoir à nouveau accéder au *shell* : la commande a été lancée au *premier plan*.

Il y a des situations, cependant, où cela n'est pas souhaitable, par exemple, lorsqu'on copie récursivement un gros répertoire vers un autre en ignorant les erreurs (rediriger le stderr vers /dev/null).

```
$ cp -R images/ /backup/ 2>/dev/null
```

Une telle commande peut prendre plusieurs minutes avant de se terminer. Si on a besoin du terminal entre temps, deux solutions sont possibles.

La première, brutale, est de tuer la commande pour la relancer plus tard. Pour ce faire, tapez [Ctrl+C]. Cela termine le processus et vous retournez alors à l'invite de commande.

La seconde est de placer le processus correspondant à cette commande en arrière-plan. Pour ce faire, il suffit de faire [Ctrl+Z] pour suspendre le processus et revenir à l'invite de commande :

```
$ cp -R images / /backup/ 2>/dev/null
# Tapez Ctrl+Z ici
[1]+  Stopped                  cp -R labos2/ /backup/ 2>/dev/null
$
```

Le processus est alors arrêté dans l'attente d'être relancé (Stopped). Il faut maintenant le relancer, mais en le maintenant en arrière-plan via la commande *bg* (*back-ground*) :

```
$ bg
[1]+ cp -R images / /backup/ 2>/dev/null &
$
```

Le processus aura alors repris son exécution en tâche de fond, ce qu'indique le signe & (esperluette) à la fin de la ligne. L'invite de commande est à nouveau disponible.

On peut également lancer la commande directement en tâche de fond en la faisant suivre de & :

```
$ cp -R images/ /backup/ 2>/dev/null &
```

Un processus qui tourne en tâche de fond, ou en arrière-plan, est appelé un *job*.

Pour constater que le processus est lancé en tâche de fond, utilisez la commande *jobs*.

```
jobs [-lnprs] [pid | nom]
```

<b>Signification</b>	Afficher la liste des processus fonctionnant en tâche de fond	
----------------------	---	--

<b>Options</b>	-l	Lister les <i>pids</i> en plus des autres informations
	-n	Lister les processus dont le statut a changé depuis le dernier rafraichissement
	-p	Lister uniquement les <i>pids</i>
	-r	Limiter la liste aux jobs dans l'état 'running'
	-s	Limiter la liste aux jobs dans l'état 'stopped'

Pour vérifier que votre processus en arrière-plan est bien actif, utilisez la commande *ps*. Si ce n'est pas le cas, envoyez-lui un signal au moyen de la commande suivante :

```
$ kill -CONT pid
```

La notion d'arrière-plan n'a un sens que sur une même console. Si vous ouvrez une autre console le processus se trouve naturellement en tâche de fond pour cette autre console...

Pour remettre le job en avant plan, il faut utiliser la commande *fg* (*foreground*).

### Quelques commandes à la suite...

Pour tester ces quelques commandes, entrez d'abord la commande suivante :

```
$ sleep 5
```

Que fait cette commande ? Le manuel vous aidera à répondre à cette question !

Entrez la même commande, avec un délai de 50, et faites en sorte qu'elle s'exécute en *background* dès son lancement.

Exécutez de nouveau cette commande, sans attendre la fin de la précédente, avec un délai de 100, en *foreground*. Stoppez-là !

Affichez la liste des processus en *background*.

Affichez la liste de tous les processus en "long" et notez la colonne statut... pour rappel : **R**unnable (en cours d'exécution), **S**leeping, **T**raced (arrêté ou suivi) ...

Faites en sorte de faire tourner la dernière commande lancée en *background*.

Affichez la liste des processus en *background* et notez la différence d'état entre les deux affichages...

Utilisez la commande `kill` pour tuer le processus exécutant la commande `sleep 100`.

La suite de ces manipulations est visible sur la figure 11.

```
piroc@svr24:~$ sleep 50 &
[1] 7439
piroc@svr24:~$ sleep 100
^Z
[2]+  Stoppé                sleep 100
piroc@svr24:~$ jobs
[1]-  En cours d'exécution  sleep 50 &
[2]+  Stoppé                sleep 100
piroc@svr24:~$ ps -l
 F S   UID     PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
 0 S   1002    7428   7427  0  80   0 -  1619 wait  pts/0    00:00:00 bash
 0 S   1002    7439   7428  0  80   0 -   936 hrtime pts/0    00:00:00 sleep
 0 T   1002    7440   7428  0  80   0 -   936 signal pts/0    00:00:00 sleep
 0 R   1002    7441   7428  0  80   0 -  1134 -      pts/0    00:00:00 ps
piroc@svr24:~$ bg
[2]+ sleep 100 &
piroc@svr24:~$ jobs
[1]-  Fini                  sleep 50
[2]+  En cours d'exécution  sleep 100 &
piroc@svr24:~$ ps
  PID TTY          TIME CMD
  7428 pts/0    00:00:00 bash
  7440 pts/0    00:00:00 sleep
  7442 pts/0    00:00:00 ps
piroc@svr24:~$ kill -KILL 7440
piroc@svr24:~$ ps
  PID TTY          TIME CMD
  7428 pts/0    00:00:00 bash
  7443 pts/0    00:00:00 ps
[2]+  Processus arrêté      sleep 100
```

Figure 11 - Exemples d'utilisation de `jobs`, `bg` et `fg`

### Exercice

Faire un programme père qui lance deux fils.

Chaque processus commence par afficher un message de début et se termine par un message de fin.

Les fils attendront quelques secondes avant d'afficher leur message de fin.

Le père attend la fin de l'exécution de ses deux fils avant d'afficher son message de fin.

Utiliser dans cet exercice les appels `fork`, `getpid`, `wait` et `sleep`.

Lancez ce programme en tâche de fond en redirigeant le `stdout` vers un fichier. Vérifiez que vos processus sont en attente avec `ps`. Tuez une tâche avec `kill`.