
Gestion des processus

Pascal Picard

Corto E.T.F., K&M

Sainte-Clotilde, Ile de la Réunion

Ivan Kurzweg

Fremens Inst.

La Possession, Ile de la Réunion

Copyright © 2002-2005, 2006 Pascal PICARD, pascal@seth.homeunix.net

Permission to use, copy, modify, and distribute this documentation for any purpose with or without fee is here by granted, provided that the above copyright notice and this permission notice appear in all copies.

THE DOCUMENTATION IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS DOCUMENTATION INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS DOCUMENTATION.

- 1. Notion de processus
- 2. Principaux éléments d'un processus
- 3. Etats d'un processus
- 4. Signaux
- 5. Commandes de gestion des processus et de leur environnement
 - 5.1. Résumé des commandes externes
 - 5.2. La commande **ps**
 - 5.3. La commande **top**
 - 5.4. Résumé des commandes internes
- 6. Tâches périodiques
- 7. Exercices
- Références

Résumé

Dans cette partie nous nous intéressons à la gestion des processus en rappelant cette notion système fondamentale. Nous étudions ensuite les principales commandes de gestion de ces processus.

Pour nous joindre : <>

<pascal@seth.homeunix.net>

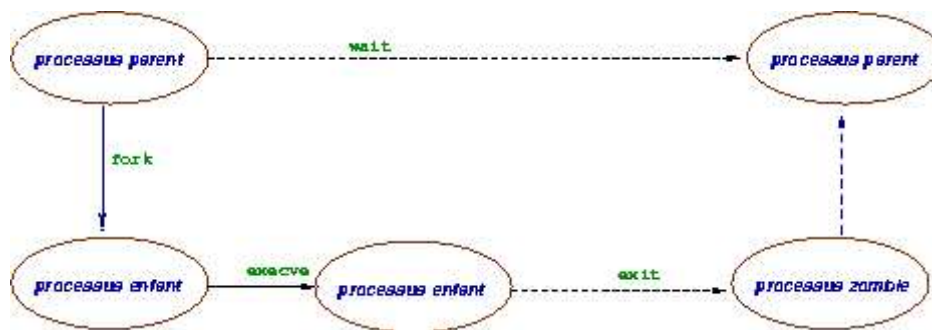
<ik-r@wanadoo.fr>

1. Notion de processus

Un processus, au sens d'un système d'exploitation est une unité logique en cours d'exécution ou susceptible de s'exécuter. De manière informelle, on dit qu'un processus est l'instance d'un programme en cours d'exécution. Durant tout son cycle de vie, qui le fait transiter par différents états (en exécution, endormi, éligible, préempté ...), le processus est identifié de manière unique au moyen d'un entier non signé appelé *pid* [*Process IDentifier*].

Un processus peut donner naissance à un nouveau processus par duplication de son contexte au moyen d'un appel système **fork** (c.f. **man 2 fork**). Cette fonction retourne deux valeurs, l'une dans le contexte du père ou elle désigne le *pid* du fils (nouvellement créé) et l'autre dans le contexte du fils (valeur égale à 0). Le processus fils partage toutes les ressources du père telles que les descripteurs de fichiers, les différents statuts de gestion des signaux, l'organisation de la mémoire. Dans la plupart des cas le processus fils est créé dans le but de charger et d'exécuter un programme différent. Le processus fils se recouvre lui-même avec l'image mémoire du nouveau programme en lui passant un ensemble de paramètres au moyen de l'appel système **execve**. Un processus se termine par l'exécution de l'appel système **exit** en passant un octet de statut à son père.

Figure 1. Cycle de vie d'un processus



Un processus peut aussi communiquer avec un autre au moyen d'un canal de communication inter-processus tel que :

- tubes [pipes],
- sockets,
- fichiers intermédiaires.

Un processus peut suspendre son exécution jusqu'à ce que tous ces fils se terminent au moyen de l'appel système **wait**, lequel renvoie le *pid* et le statut du fils terminé. Le processus père peut s'arranger pour recevoir une notification par un signal à la terminaison (correcte ou incorrecte) du processus fils.

Un processus fils peut devenir orphelin si son père termine avant lui, auquel cas le noyau s'arrange pour le « faire adopter » par un processus système (*INIT*), le processus fils peut donc lui transmettre son statut de terminaison.

L'exécution des processus est géré par un (processus) *planificateur* (ou ordonnanceur [*scheduler*]) selon la priorité attribué aux processus. Cette priorité est gérée par un *algorithme d'ordonnement* [*scheduling algorithm*] du noyau.

2. Principaux éléments d'un processus

La plupart des éléments d'un processus (temps *CPU* consommé, fichiers ouverts et accédés, mémoire allouée ...) ont une influence sur son exécution. Nous abordons dans cette section les éléments ou paramètres les plus intéressants du point de vue de l'administrateur système.

pid

Il s'agit de l'identifiant du processus assigné par le noyau. Son unicité permet de repérer le processus sans ambiguïtés sur tout son cycle de vie. Le noyau attribue les numéros de processus dans l'ordre de leur apparition. Quand il a épuisé son intervalle de valeurs, il repart sur la borne inférieure en sautant les valeurs attribuées aux processus en cours.

ppid

C'est l'identifiant du processus père du processus courant. Un processus est créé par duplication du contexte d'un processus existant. Le processeur géniteur est dit père et le processus nouvellement créé est dit fils. Tout processus possède un père unique et un père peut avoir différents fils (arborescence de processus). Si un processus père vient à terminer avant son fils, ce dernier hérite d'un nouveau père le processus *INIT*.

uid et *euid*

L'*uid* [*user identifier*] désigne l'identifiant de l'utilisateur propriétaire du processus (i.e. l'identité de l'utilisateur ayant créé le processus). Plus précisément, il s'agit de l'*euid* [*effective user identifier*] associé au processus père du processus courant.

L'*euid* est utilisé pour déterminer les ressources (mémoires, fichiers, ...) auxquelles un processus peut accéder. Dans la plupart des cas l'*euid* est égal à l'*uid*, sauf si le programme exécuté est *setuid on execution*.

Conserver ces deux valeurs permet de maintenir la distinction entre identité et permissions, et autorise le programme *setuid* à ne pas s'exécuter sur tout son cycle de vie en usant de permissions étendues, i.e. l'*euid* évolue en fonction des permissions nécessaires.

Les Unices conservent en général une autre valeur appelée *suid* [*save uid*], copie de l'*euid* au moment du début de son exécution. Le programme *setuid* peut ainsi renoncer à ces privilèges d'accès durant la majeure partie de son exécution et retrouver ses privilèges uniquement quand cela est nécessaire. Cela rend le programme plus sûr du point de vue sécurité.

gid et *egid*

Le *gid* [*group identifier*] désigne l'identifiant du groupe propriétaire du processus. L'*egid* [*effective group identifier*] entretient avec le *gid* une relation similaire à celle liant le couple *euid*, *uid*. Un processus peut appartenir à plusieurs groupes à la fois. La détermination des permissions d'accès dépend uniquement de *egid* et de la liste des groupes supplémentaires (stockés indépendamment de l'*egid* et du *gid*).

nice

La priorité d'ordonnement d'un processus est déterminée par un algorithme d'ordonnement qui se base sur des paramètres tels que le temps *CPU* récemment consommé, le temps d'attente avant exécution et le paramètre *nice* ajustable par l'utilisateur. Sur les architectures *BSD* *nice* est défini dans l'intervalle [-20, +20] ([-20, +19] sur *GNU Linux*), une valeur négative augmente la priorité du processus, tandis qu'une valeur positive la diminue. Seul l'administrateur est en mesure d'attribuer des valeurs négatives.

Control terminal

La plupart des processus possèdent un terminal de contrôle qui leur permet notamment d'ajuster leur entrée standard et leurs sorties standard et d'erreurs.

3. Etats d'un processus

A un instant donné, sur une machine mono-processeur, un seul processus peut être exécuté (et au plus n sur une machine à n processeurs). En général, le nombre m de processus existant est bien supérieur au nombre n de processeurs, par conséquent les processus possèdent plusieurs états (autre que l'état « en exécution ». Il existe fondamentalement quatre états d'exécution :

Tableau 1. Les principaux états d'un processus

Etats	Sémantique
<i>Runnable</i>	Prêt à s'exécuter.
<i>Sleeping</i>	En attente d'un évènement ou d'une ressource.
<i>Stopped</i>	Suspendu.
<i>Zombie</i>	Etat terminal d'un processus.

Un processus dans l'état *Runnable* est prêt à s'exécuter dès que le *CPU* est disponible. Trois cas sont possibles :

- Le processus s'exécute pendant tout le **quantum** qui lui est alloué avant commutation.
- Il est *préempté* suite à un appel système (le processus demande une opération d'E/S, par exemple)
- Il termine son travail avant l'expiration de ce **quantum**, un autre processus est alors exécuté.

Dans les deux premiers cas il passe dans l'état *Sleeping*, tandis que dans le dernier il passe dans l'état *Zombie*, tant qu'il n'a pas transmis son statut de terminaison à son père.

Un processus dans l'état « endormi » [*Sleeping*] est dans l'attente d'un évènement. Par exemple, fin d'une opération d'E/S, demande de connexion réseau à un démon. La fin de l'évènement est marqué par un signal qui permet au processus de redevenir éligible (état *Runnable*).

L'état *Stopped* interdit au processus de s'exécuter. Un processus passe dans cet état suite à l'envoi du signal *SIGSTOP* ou par *SIGTSTP* (généré par un <CTRL>-Z, c.f. section Section 4, « Signaux »). La reprise est assurée à la réception du signal *SIGCONT*.

L'état *Zombie* marque la fin de l'exécution d'un processus, qui n'a pas encore transmit son statut de terminaison à son père.

4. Signaux

Il s'agit d'un mécanisme basique de communication unidirectionnel avec les processus. Les différents « *Unices* » définissent près d'une trentaine de signaux.

Les signaux peuvent être envoyé par un utilisateur par la commande **kill**, généré depuis un terminal pour interrompre ((ou terminer) <CTRL>-C) un processus, ou pour suspendre (<CTRL>-Z) un processus, ou envoyer par le noyau pour signaler un problème (division par 0, par exemple).

A la réception d'un signal, un processus peut :

- activer un gestionnaire [*handler*] prédéfini pour traiter l'évènement, ou
- laisser au noyau le soin d'exécuter une action par défaut, laquelle dépend de la nature du signal.

Un processus n'est pas toujours en mesure de traiter immédiatement un signal quand celui-ci survient, c'est pourquoi deux mécanismes supplémentaires sont prévus :

- *ignorer* le signal, i.e. ne pas en tenir compte.
- *suspendre* ou *bloquer* le signal, i.e. le mettre en attente jusqu'à ce que le processus soit en mesure de le traiter. Dans ce cas le gestionnaire associé au signal n'est appelé qu'une seule fois, même si le signal a été délivré plusieurs fois pendant la durée de la suspension.

Récapitulatif des signaux de base.

Tableau 2. Commandes externes

n°	nom	description	action par défaut	ignorable ?	suspendable ?	core dump ?
1	<i>SIGHUP</i>	<i>terminal line hangup</i>	Terminaison	Oui	Oui	Non
2	<i>SIGINT</i>	<i>interrupt program</i>	Terminaison	Oui	Oui	Non
3	<i>SIGQUIT</i>	<i>quit program</i>	Terminaison	Oui	Oui	Oui
9	<i>SIGKILL</i>	<i>kill program</i>	Terminaison inconditionnelle	Non	Non	Non
*	<i>SIGBUS</i>	bus error	Terminaison	Oui	Oui	Oui
11	<i>SIGSEGV</i>	<i>Segmentation violation</i>	Terminaison	Oui	Oui	Oui
15	<i>SIGTERM</i>	<i>Software termination</i>	Terminaison	Oui	Oui	Non
*	<i>SIGSTOP</i>	<i>STOP</i>	Interruption inconditionnelle	Non	Non	Non

n°	nom	description	action par défaut	ignorable ?	suspendable ?	core dump ?
*	<i>SIGTSTP</i>	<i>Terminal stop</i>	Interruption générée par un clavier	Oui	Oui	Non
*	<i>SIGCONT</i>	<i>Continue after stop</i>	Ignorer	Non	Non	Non
*	<i>SIGWINCH</i>	<i>Window size change</i>	Ignorer	Oui	Oui	Non
*	<i>SIGUSR1</i>	<i>User defined</i>	Terminaison	Oui	Oui	Non
*	<i>SIGUSR2</i>	<i>User defined</i>	Terminaison	Oui	Oui	Non

Il existe d'autres signaux (c.f. **man 3 signal**, sous *FreeBSD*).

Les signaux *SIGKILL* et *SIGSTOP* ne peuvent être ni géré par un gestionnaire, ni ignoré, ni suspendu (bloqué). Le premier détruit (i.e. terminaison brutale) le processus cible (du signal), tandis que le second suspend l'exécution du processus cible jusqu'à réception du signal symétrique *SIGCONT*. Ce dernier peut quant à lui être géré par un gestionnaire ou ignoré mais ne peut pas être bloqué.

Les signaux *SIGHUP*, *SIGINT*, *SIGQUIT* et *SIGTERM* ont la signification suivante :

- *SIGHUP* : possèdent deux interprétations. La première permet à un démon (service) s'il est capable de relire son/ses fichier(s) de configuration, de se recharger (suite à des modifications) sans avoir à le redémarrer. La seconde, permet de terminer (« tuer ») un processus attaché à un terminal, i.e. quitter son terminal revient à terminer les processus qui lui sont attachés.

Les processus asynchrones des shells de la famille *cs*h et *tc*sh sont immunisés par défaut contre ce signal. Pour ceux de la famille des shells *Bourne* (*ksh*, *zsh*, *bash* ...) cette immunité peut être simulée avec la commande **nohup**.

- *SIGINT* : ce signal est émis en tapant un <CTRL-C>. Il a pour effet de terminer (correctement) le processus courant.
- *SIGQUIT* : entraîne la terminaison correcte du processus cible en produisant un fichier image (*aka core dump*) à moins qu'il ne soit intercepté par un gestionnaire de signal.
- *SIGTERM* : entraîne la terminaison correcte du processus cible.

Les signaux *SIGUSR1* et *SIGUSR2* n'ont pas de sémantique précise. Ils sont éventuellement utilisés par les programmes qui leur attribuent librement une signification.

Envoi de signaux avec KILL (FreeBSD). Cette commande permet à l'utilisateur d'envoyer des signaux aux processus dont il est propriétaire. Cette restriction ne s'applique évidemment pas au *super*-utilisateur.

SYNOPSIS :

```
kill [-s signal_name] pid
-l [exit_status]
-signal_name pid...
-signal_number pid...
```

EXEMPLE : 1) Terminer les processus de pid 1066 et 1072

```
pascal@Godel:~ > kill 1066 1072
```

EXEMPLE : 2) Envoyer le signal *SIGHUP* au processus de pid 657

```
pascal@Godel:~ > kill -s HUP 657
```

EXEMPLE : 3) Envoyer le signal *SIGHUP* au processus de pid 657 (alternative)

```
pascal@Godel:~ > kill -HUP 657
```

Sans numéro ou nom de signal explicite, la commande **kill** envoie le signal *SIGTERM* au processus cible. Il n'y a donc pas de garantie de terminaison puisque ce signal peut être intercepté (par un gestionnaire), ignoré ou bloqué.

Envoi de signaux avec KILL (GNU/Linux). Elle fonctionne de manière similaire à son alter-ego *FreeBSD*, consulter le **man 3 signal**.

Envoi de signaux avec KILLALL (FreeBSD). Si on ne connaît pas le *pid* du processus auquel on veut envoyer un signal, on peut soit le chercher via la commande **ps**, ou bien utiliser la commande **killall** pour faire ce travail. Dans ce cas encore, il faut être propriétaire du processus.

SYNOPSIS :

```
killall [-d | -v ] [-h | -? ] [-help] [-l] [-m] [-s] [-u user] [-t tty]
[-c procname] [-SIGNAL] [procname...]
```

EXEMPLE : Terminer le/les processus de nom *mozilla-bin*

```
pascal@Godel:~ > killall mozilla-bin
```

5. Commandes de gestion des processus et de leur environnement

5.1. Résumé des commandes externes

Tableau 3. Commandes externes

Commandes	Sémantique
ps	Cliché à un instant donné des processus en cours d'exécution
top	Etat en temps <i>réel</i> des processus en cours d'exécution.
kill	Utilitaire permettant l'envoi de signal à un processus par l'intermédiaire de son <i>pid</i> .
killall	Utilitaire permettant d'envoyer un signal de terminaison (<i>SIGTERM</i>) à des processus, repérés par leur nom.
su	c.f. commande ???
ident	Affiche toutes les chaînes d'identification <i>RCS</i> d'un fichier, i.e. les informations sur la version d'une commande.
fuser	(<i>GNU/Linux</i>) Liste les processus utilisant un fichier (ou un disque).
lsuf	En l'absence d'arguments, cette commande liste tous les fichiers ouverts de tous les processus en cours.
at	Permet d'exécuter des commandes en temps différé.
crontab	Permet d'exécuter des commandes périodiquement.
jobs	Liste les jobs en cours.
<CTRL>-C	Envoit un signal de terminaison (<i>SIGTERM</i>) au processus synchrone courant.
<CTRL>-Z	Envoit un signal de suspension (<i>SIGSTOP</i>) au processus synchrone courant.
suspend	Suspend un shell (signal <i>SIGSTOP</i>), souvent utilisé pour suspendre un shell lancé par la commande su .
env	Affiche l'environnement courant ou bien le modifie ou bien le crée.
printenv	Permet de visualiser l'environnement courant.
nice	Permet de modifier la priorité d'un processus lors de son lancement. Seul le <i>super</i> -utilisateur est en mesure de baisser la valeur, donc d'augmenter la priorité d'un processus (valeur relative).
renice	Permet de modifier la priorité d'un processus en cours d'exécution, attention on donne une valeur absolue.

5.2. La commande ps

C'est la commande principale d'audit des processus. Elle permet d'obtenir un cliché à un instant donné de l'état du système (c.f. **man 1 ps**). L'implémentation *GNU* de cette commande lui permet d'accepter à la fois la syntaxe *BSD* et *System V*, sur un système *GNU Linux*.

Les principales options d'un système *BSD* sont :

- **a** : affiche les processus des autres utilisateurs.
- **l** : affiche des informations supplémentaires associées aux mots clés suivants : *pid, ppid, cpu, pri, nice, vsz, rss, wchan, state, tt, time* et *command*.
- **u** : affiche des informations associés au mots clés suivants : *user, pid, %cpu, %mem, vsz, rss, tt, state, start, time* et *command*.
- **U** *username* : affiche les processus appartenant à *username*.
- **w** : affichage en mode 132 colonnes.
- **x** : affiche des informations sur les processus ne possédant pas de terminal de controle.

Exemples

Orientation *BSD*, lister tous les processus :

```
pascal@Godel:~ > ps aux
USER      PID %CPU %MEM    VSZ   RSS  TT  STAT  STARTED      TIME COMMAND
pascal   12018  0.0  0.0   460   256 p4  R+    2:56PM    0:00.00 ps aux
root         1  0.0  0.0   552   316 ??  ILs   Sun03PM   0:00.04 /sbin/init --
root         2  0.0  0.0     0     0 ??  DL    Sun03PM   0:00.59 (pagedaemon)
root         3  0.0  0.0     0     0 ??  DL    Sun03PM   0:00.00 (vmdaemon)
root         4  0.0  0.0     0     0 ??  DL    Sun03PM   0:02.99 (bufdaemon)
root         5  0.0  0.0     0     0 ??  DL    Sun03PM   0:02.59 (vnlru)
root         6  0.0  0.0     0     0 ??  DL    Sun03PM   3:11.79 (syncer)
root        12  0.0  0.0   592   288 ??  DLs   Sun03PM   0:00.00 vinum: vinum daemon (vinum)
root        36  0.0  0.0   212    96 ??  Is    Sun03PM   0:00.00 adjkerntz -i
root        90  0.0  0.1   944   588 ??  Ss    Sun11AM   0:00.96 /usr/sbin/syslogd -ss
root        98  0.0  0.1   988   724 ??  Ss    Sun11AM   0:01.01 /usr/sbin/cron
root       100  0.0  0.2  2320  1644 ??  Ss    Sun11AM   0:00.06 /usr/sbin/sshd
```

root	216	0.0	0.3	3620	3116	??	Ss	Sun11AM	8:17.45	/usr/local/sbin/cupsd
root	294	0.0	0.1	948	660	v0	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv0
root	295	0.0	0.1	948	660	v1	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv1
root	296	0.0	0.1	948	660	v2	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv2
root	297	0.0	0.1	948	660	v3	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv3
root	298	0.0	0.1	948	660	v4	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv4
root	299	0.0	0.1	948	660	v5	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv5
root	300	0.0	0.1	948	660	v6	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv6
root	301	0.0	0.1	948	660	v7	Is+	Sun11AM	0:00.00	/usr/libexec/getty Pc ttyv7
root	302	0.0	0.2	4436	2052	??	S	Sun11AM	0:00.09	/usr/X11R6/bin/wdm -nodaemon ttyv8
pascal	335	0.0	0.4	5468	4136	??	Ss	Sun11AM	1:31.67	/usr/X11R6/bin/wmaker
pascal	350	0.0	0.1	1928	1404	??	Ss	Sun11AM	0:02.46	ssh-agent /usr/X11R6/bin/wmaker
pascal	354	0.0	0.5	6108	5124	??	S	Sun11AM	0:04.25	Eterm -X /home/pascal/.Eterm/MAIN
pascal	358	1.0	1.1	13624	11360	??	S	Sun11AM	71:30.17	gkrellm
pascal	359	0.0	0.5	6612	5652	??	S	Sun11AM	0:04.92	Eterm -X /home/pascal/.Eterm/MAIN
pascal	362	0.0	0.1	2104	1524	??	S	Sun11AM	2:08.34	/usr/X11R6/bin/wmitime
pascal	363	0.0	0.1	2052	1440	??	S	Sun11AM	6:36.14	wmmatrix -med
pascal	399	0.0	0.2	3236	1648	??	Ss	Sun11AM	0:03.61	/usr/local/bin/fetchmail -F -K
pascal	413	0.0	0.2	2840	2248	??	S	Sun11AM	0:08.34	/usr/X11R6/bin/xscreensaver
pascal	437	0.0	0.2	2048	1780	p4	Ss	Sun11AM	0:00.69	-tcsh (tcsh)
pascal	467	0.0	0.2	2056	1780	p5	Is	Sun11AM	0:00.25	-tcsh (tcsh)
pascal	607	0.0	0.3	4188	3252	p5	S+	Sun11AM	0:08.20	mutt
pascal	1782	0.0	0.2	2080	1712	p0	Ss+	Sun08PM	0:00.28	ssh -p 3322 -v pascal@Turing
pascal	10556	0.0	1.1	13256	11568	p4	S	8:12AM	1:06.67	xemacs p2_proc.xml (xemacs-21)
pascal	11320	0.0	1.1	16804	11620	??	S	10:55AM	5:38.90	/usr/X11R6/bin/xmms
root	0	0.0	0.0	0	0	??	DLs	Sun03PM	0:00.00	(swapper)

Lister les processus de l'utilisateur *pascal* :

```
pascal@Godel:~ > ps U pascal
  PID  TT  STAT      TIME COMMAND
  335  ??  Ss      1:32.48 /usr/X11R6/bin/wmaker
  350  ??  Ss      0:02.47 ssh-agent /usr/X11R6/bin/wmaker
  354  ??  S       0:04.27 Eterm -X /home/pascal/.Eterm/MAIN
  358  ??  S      71:57.96 gkrellm
  359  ??  S       0:05.40 Eterm -X /home/pascal/.Eterm/MAIN
  362  ??  S       2:08.96 /usr/X11R6/bin/wmitime
  363  ??  S       6:38.70 wmmatrix -med
  399  ??  Is      0:03.61 /usr/local/bin/fetchmail -F -K
  413  ??  S       0:08.53 /usr/X11R6/bin/xscreensaver
11320 ??  S       5:57.42 /usr/X11R6/bin/xmms
11840 ??  S       0:00.31 Eterm -X /home/pascal/.Eterm/MAIN
 1782 p0  Is+     0:00.28 ssh -p 3322 -v pascal@Turing.corto.home
  437 p4  Is      0:00.23 -tcsh (tcsh)
10556 p4  S       1:09.24 xemacs p2_proc.xml (xemacs-21)
12130 p4  R+      0:00.00 ps U pascal
  467 p5  Is      0:00.25 -tcsh (tcsh)
  607 p5  I+      0:08.20 mutt
```

Les principales options d'un système *GNU Linux* sont :

Orientation *System V* :

- `-e` : affiche des informations sur tous les processus.
- `-f` : affiche des informations supplémentaires associées aux mots clés suivants : *uid, pid, ppid, c, stime* et *command*.
- `-u username` : affiche les informations relatives à tous les processus de l'utilisateur *username*.

Exemples

Orienté *System V*

```
pascal@tux:~ > ps -ef
  UID      PID  PPID  C  STIME  TTY          TIME CMD
  root         1    0  0 14:19  ?           00:00:03 init [2]
  root         2    1  0 14:19  ?           00:00:00 [keventd]
  root         3    0  0 14:19  ?           00:00:00 [ksoftirqd_CPU0]
  root         4    0  0 14:19  ?           00:00:00 [kswapd]
  root         5    0  0 14:19  ?           00:00:00 [bdflush]
  root         6    0  0 14:19  ?           00:00:00 [kupdated]
  root         7    1  0 14:19  ?           00:00:00 [kjournald]
  root        42    1  0 14:19  ?           00:00:00 [khud]
  root        82    1  0 14:19  ?           00:00:00 [kjournald]
  root        83    1  0 14:19  ?           00:00:00 [kjournald]
  root        84    1  0 14:19  ?           00:00:00 [kjournald]
  root        85    1  0 14:19  ?           00:00:00 [kjournald]
  root        86    1  0 14:19  ?           00:00:00 [kjournald]
  root        87    1  0 14:19  ?           00:00:00 [kjournald]
  root        88    1  0 14:19  ?           00:00:00 [kjournald]
  root       114    1  0 14:20  ?           00:00:00 [eth0]
  root       195    1  0 14:20  ?           00:00:00 /sbin/syslogd
  root       198    1  0 14:20  ?           00:00:00 /sbin/klogd
  root       366    1  0 14:20  ?           00:00:00 /usr/sbin/sshd
 daemon     381    1  0 14:20  ?           00:00:00 /usr/sbin/atd
```

```

root      384      1  0 14:20 ?          00:00:00 /usr/sbin/cron
root      398      1  0 14:20 ?          00:00:00 /usr/sbin/apache-ssl
www-data  408     398  0 14:20 ?          00:00:00 /usr/lib/apache-ssl/gcache 33 /var/run/gcache_po
www-data  412     398  0 14:20 ?          00:00:00 /usr/sbin/apache-ssl
root      417      1  0 14:20 ?          00:00:00 /usr/bin/X11/wdm
root      420      1  0 14:20 tty1        00:00:00 /sbin/getty 38400 tty1
root      421      1  0 14:20 tty2        00:00:00 /sbin/getty 38400 tty2
root      422      1  0 14:20 tty3        00:00:00 /sbin/getty 38400 tty3
root      423      1  0 14:20 tty4        00:00:00 /sbin/getty 38400 tty4
root      427     417  0 14:20 ?          00:00:01 /usr/bin/X11/X -nolisten TCP -auth /var/lib/wdm/
root      428     417  0 14:20 ?          00:00:00 -:0
root      437     428  0 14:20 ?          00:00:00 wdmLogin -d:0 -wdefault:twm:wmaker -l/usr/share/
root      438     366  0 14:21 ?          00:00:00 /usr/sbin/sshd
pascal   440     438  0 14:21 ?          00:00:00 /usr/sbin/sshd
pascal   441     440  0 14:21 pts/0      00:00:00 -bash
pascal   518     441  0 15:33 pts/0      00:00:00 ps -ef

```

Orienté BSD

```

pascal@tux:~ > ps aux
USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.0   1212   472 ?        S    14:19   0:03 init [2]
root         2   0.0   0.0     0     0 ?        SW   14:19   0:00 [keventd]
root         3   0.0   0.0     0     0 ?        SWN  14:19   0:00 [ksoftirqd_CPU0]
root         4   0.0   0.0     0     0 ?        SW   14:19   0:00 [kswapd]
root         5   0.0   0.0     0     0 ?        SW   14:19   0:00 [bdflush]
root         6   0.0   0.0     0     0 ?        SW   14:19   0:00 [kupdated]
root         7   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        42   0.0   0.0     0     0 ?        SW   14:19   0:00 [khubd]
root        82   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        83   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        84   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        85   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        86   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        87   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root        88   0.0   0.0     0     0 ?        SW   14:19   0:00 [kjournald]
root       114   0.0   0.0     0     0 ?        SW   14:19   0:00 [eth0]
root       195   0.0   0.1   1984   756 ?        S    14:20   0:00 /sbin/syslogd
root       198   0.0   0.2   1856  1128 ?        S    14:20   0:00 /sbin/klogd
root       366   0.0   0.2   2492  1192 ?        S    14:20   0:00 /usr/sbin/sshd
daemon     381   0.0   0.1   1316   552 ?        S    14:20   0:00 /usr/sbin/atd
root       384   0.0   0.1   1396   664 ?        S    14:20   0:00 /usr/sbin/cron
root       398   0.0   0.9  73840  4824 ?        S    14:20   0:00 /usr/sbin/apache-ssl
www-data   408   0.0   0.1   2872   784 ?        S    14:20   0:00 /usr/lib/apache-ssl/gcache 33 /va
www-data   412   0.0   0.9  73856  4848 ?        S    14:20   0:00 /usr/sbin/apache-ssl
root       417   0.0   0.2   3960  1256 ?        S    14:20   0:00 /usr/bin/X11/wdm
root       420   0.0   0.0   1200   444 tty1      S    14:20   0:00 /sbin/getty 38400 tty1
root       421   0.0   0.0   1200   444 tty2      S    14:20   0:00 /sbin/getty 38400 tty2
root       422   0.0   0.0   1200   444 tty3      S    14:20   0:00 /sbin/getty 38400 tty3
root       423   0.0   0.0   1200   444 tty4      S    14:20   0:00 /sbin/getty 38400 tty4
root       427   0.0   1.4  58320  7456 ?        S<   14:20   0:01 /usr/bin/X11/X -nolisten TCP -aut
root       428   0.0   0.2   3964  1352 ?        S    14:20   0:00 -:0
root       437   0.0   0.3   4364  2032 ?        S    14:20   0:00 wdmLogin -d:0 -wdefault:twm:wmake
root       438   0.0   0.3   6200  1844 ?        S    14:21   0:00 /usr/sbin/sshd
pascal    440   0.0   0.3   6288  1916 ?        R    14:21   0:00 /usr/sbin/sshd
pascal    441   0.0   0.2   2308  1356 pts/0    S    14:21   0:00 -bash
pascal    539   0.0   0.1   2488   752 pts/0    R    15:54   0:00 ps aux

```

visualiser la hiérarchie des processus avec l'option `f` (*forest*)

```

pascal@tux:~ > ps faux
USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.0   1208   464 ?        S    Sep10   0:04 init
root         2   0.0   0.0     0     0 ?        SW   Sep10   0:00 [keventd]
root         3   0.0   0.0     0     0 ?        SWN  Sep10   0:00 [ksoftirqd_CPU0]
root         4   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kswapd]
root         5   0.0   0.0     0     0 ?        SW   Sep10   0:00 [bdflush]
root         6   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kupdated]
root         7   0.0   0.0     0     0 ?        SW<  Sep10   0:00 [mdrecoveryd]
root         8   0.0   0.0     0     0 ?        SW<  Sep10   0:00 [raidld]
root         9   0.0   0.0     0     0 ?        SW<  Sep10   0:00 [raidld]
root        10   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        87   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        88   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        89   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        90   0.0   0.0     0     0 ?        SW   Sep10   0:01 [kjournald]
root        91   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        92   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        93   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        94   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        95   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        96   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        97   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        98   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root        99   0.0   0.0     0     0 ?        SW   Sep10   0:00 [kjournald]
root       192   0.0   0.2   1832  1104 ?        S    Sep10   0:00 /sbin/klogd

```

root	203	0.0	0.2	2312	1212	?	S	Sep10	0:00	/bin/sh /usr/bin/
mysql	240	0.0	1.2	38964	6304	?	S	Sep10	0:01	_ /usr/sbin/mys
mysql	244	0.0	1.2	38964	6304	?	S	Sep10	0:01	_ /usr/sbin
mysql	245	0.0	1.2	38964	6304	?	S	Sep10	0:00	_ /usr/
mysql	246	0.0	1.2	38964	6304	?	S	Sep10	0:00	_ /usr/
daemon	262	0.0	0.1	1316	552	?	S	Sep10	0:00	/usr/sbin/atd
root	265	0.0	0.1	1396	660	?	S	Sep10	0:00	/usr/sbin/cron
root	282	0.0	0.0	1200	444	tty4	S	Sep10	0:00	/sbin/getty 38400
root	283	0.0	0.0	1200	444	tty5	S	Sep10	0:00	/sbin/getty 38400
root	284	0.0	0.0	1200	444	tty6	S	Sep10	0:00	/sbin/getty 38400
root	586	0.0	0.2	2432	1492	tty1	S	Sep10	0:00	-bash
root	652	0.0	0.1	1288	576	?	S	Sep10	0:00	/sbin/syslogd -a
root	858	0.0	0.3	3864	2020	?	S	Sep10	0:00	/usr/sbin/apache
japache	859	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	860	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	861	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	862	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	863	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	864	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	865	0.0	0.4	3932	2172	?	S	Sep10	0:00	_ /usr/sbin/apa
japache	3857	0.0	0.4	3932	2148	?	S	Sep13	0:00	_ /usr/sbin/apa
root	15968	0.0	0.7	73072	3924	?	S	Sep24	0:00	/usr/sbin/apache-
japache	15969	0.0	0.2	3756	1132	?	S	Sep24	0:00	_ /usr/lib/apac
japache	27240	0.0	1.0	73732	5612	?	S	Oct02	0:01	_ /usr/sbin/apa
japache	27847	0.0	1.0	73656	5512	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27865	0.0	1.0	73660	5520	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27869	0.0	1.0	73596	5440	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27870	0.0	1.0	73640	5532	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27871	0.0	1.0	73664	5536	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27923	0.0	1.0	73580	5468	?	S	Oct02	0:00	_ /usr/sbin/apa
japache	27986	0.0	1.0	73644	5520	?	S	Oct02	0:00	_ /usr/sbin/apa
root	17691	0.0	0.0	1200	444	tty2	S	Sep25	0:00	/sbin/getty 38400
root	24280	0.0	0.2	2684	1348	?	S	Sep30	0:00	/usr/sbin/sshd
root	28525	0.0	0.3	5684	1748	?	S	06:56	0:00	_ sshd: pascal
pascal	28527	0.0	0.3	5708	1812	?	R	06:56	0:00	_ sshd: pas
pascal	28528	0.2	0.2	2408	1460	pts/0	S	06:56	0:00	_ -bash
pascal	28532	0.0	0.1	2528	752	pts/0	R	06:56	0:00	_ ps faux
root	26076	0.0	0.0	1200	444	tty3	S	Oct01	0:00	/sbin/getty 38400

5.3. La commande top

Cette commande permet d'avoir des informations en temps réel sur les processus.

```
pascal@Godel:~ > top
last pid: 12503; load averages: 0.19, 0.09, 0.08 up 4+04:56:36 16:22:33
38 processes: 1 running, 37 sleeping
CPU states: 7.0% user, 0.0% nice, 3.5% system, 1.2% interrupt, 88.4% idle
Mem: 150M Active, 498M Inact, 76M Wired, 132K Cache, 112M Buf, 279M Free
Swap: 4096M Total, 4096M Free

  PID USERNAME PRI NICE  SIZE  RES STATE   TIME  WCPU   CPU COMMAND
  308 root          2  0  144M  141M select 95:19  1.95%  1.95% XFree86
  359 pascal        2  0  6872K 5912K select 0:08  1.17%  1.17% Eterm
  358 pascal        2  0 13624K 11360K poll 73:04  1.03%  1.03% gkrellm
12503 pascal       28  0  1992K  1208K RUN    0:00  1.75%  0.63% top
11320 pascal        2  0 16804K 11620K poll  6:40  0.10%  0.10% xmms
  216 root          2  0  3620K  3116K select 8:18  0.00%  0.00% cupsd
  363 pascal        2  0  2052K  1440K select 6:45  0.00%  0.00% wmmatrix
  767 pascal        2  0 64436K 57352K poll  4:10  0.00%  0.00% mozilla-bin
  362 pascal       10  0  2104K  1524K nanslp 2:10  0.00%  0.00% wmitime
  335 pascal        2  0  5468K  4136K select 1:35  0.00%  0.00% wmaker
10556 pascal        2  0 13324K 12564K poll  1:31  0.00%  0.00% xemacs-21.1.14
  413 pascal        2  0  2840K  2248K poll  0:09  0.00%  0.00% xscreensaver
  320 nobody       10  0  2288K  1740K nanslp 0:09  0.00%  0.00% cups-pollD
 5051 pascal        2  0  6072K  4768K poll  0:07  0.00%  0.00% xdiary
  354 pascal        2  0  6108K  5124K select 0:04  0.00%  0.00% Eterm
  399 pascal        2  0  3236K  1648K select 0:04  0.00%  0.00% fetchmail
  350 pascal        2  0  1928K  1404K select 0:02  0.00%  0.00% ssh-agent
 1781 pascal        2  0  2368K  1784K select 0:02  0.00%  0.00% rxvt
11840 pascal        2  0  6680K  5844K select 0:02  0.00%  0.00% Eterm
   98 root         10  0   988K   724K nanslp 0:01  0.00%  0.00% cron
   90 root          2  0   944K   588K select 0:01  0.00%  0.00% syslogd
  437 pascal       18  0  2048K  1792K pause 0:01  0.00%  0.00% tcsh
 1782 pascal        2  0  2080K  1712K select 0:00  0.00%  0.00% ssh
11860 pascal        2  0  2204K  1736K select 0:00  0.00%  0.00% ssh
  467 pascal       18  0  2056K  1780K pause 0:00  0.00%  0.00% tcsh
11843 pascal       18  0  2056K  1776K pause 0:00  0.00%  0.00% tcsh
  302 root          2  0  4436K  2052K select 0:00  0.00%  0.00% wdm
  100 root          2  0  2320K  1644K select 0:00  0.00%  0.00% sshd
12450 pascal        2  0  2824K  1908K poll  0:00  0.00%  0.00% mutt
  309 root         10  0  4544K  2308K wait  0:00  0.00%  0.00% wdm
  258 root         10  0   644K   452K wait  0:00  0.00%  0.00% sh
  756 pascal       10  0   652K   460K wait  0:00  0.00%  0.00% sh
  294 root          3  0   948K   660K ttyin  0:00  0.00%  0.00% getty
  297 root          3  0   948K   660K ttyin  0:00  0.00%  0.00% getty
  300 root          3  0   948K   660K ttyin  0:00  0.00%  0.00% getty
```

296	root	3	0	948K	660K	ttyin	0:00	0.00%	0.00%	getty
301	root	3	0	948K	660K	ttyin	0:00	0.00%	0.00%	getty
36	root	18	0	212K	96K	pause	0:00	0.00%	0.00%	adjkerntz

Par défaut, l'affichage est rafraîchi toutes les 2 s sur un système *FreeBSD* (tous les 5 s sur un système *Debian*), les processus les plus actifs apparaissant au sommet de la liste. Cette commande prend en compte les entrées du terminal courant, il est donc possible d'envoyer des signaux aux processus et/ou de modifier leur priorité (**renice**) ou encore de modifier l'ordre d'affichage.

Attention cette commande consomme un temps *CPU* non négligeable, son usage est donc à réserver à des fins de diagnostics.

5.4. Résumé des commandes internes

Tableau 4. Commandes internes

Commandes	Sémantique
<cmd> &	Lance l'exécution de la commande <cmd> de manière asynchrone (arrière-plan).
<cmd>	Lance l'exécution de la commande de manière synchrone (avant-plan).
exec <cmd>	Substitut au shell courant, l'exécution de la commande <cmd>.
wait	Attend la terminaison de tous les processus asynchrones enfants du processus courant.
. <cmd> ou source <cmd>	Exécute la commande <cmd> dans le contexte du shell courant, i.e. pas de fork .

6. Tâches périodiques

Le démon **crond** est chargé d'exécuter les processus (ou tâches) régulières sur les systèmes *Unix*, tandis que le démon **atd** exécute des processus différés dans le temps.

L'administrateur dispose de deux fichiers pour définir qui est autorisé à exécuter des processus périodiques : `/var/cron/allow`, `/var/cron/deny` sous *FreeBSD* (resp. dans `/etc/cron.allow`, `/etc/cron.deny` sous *GNU Linux*)

Si ces fichiers n'existent pas alors deux cas symétriques, selon les systèmes :

1. seul « root » est autorisé ou
2. tous les utilisateurs sont autorisés (cas par défaut pour *FreeBSD* et *Debian*).

Si le fichier `/var/cron/allow` (`/etc/cron.allow`), alors les utilisateurs qui y figurent (à raison d'un par ligne) sont autorisés à exécuter des tâches « cron », les autres non. Autrement dit, on interdit par défaut tous les utilisateurs et on explicite les autorisations.

On a le cas symétrique si le fichier `/var/cron/deny` (`/etc/cron.deny`) existe, i.e. les utilisateurs qui y figurent (à raison d'un par ligne) ne sont pas autorisés à exécuter des tâches « cron », les autres oui.

Si on figure dans les deux fichiers, l'autorisation l'emporte !

Le démon **crond** enregistre son activité dans le fichier `/var/log/cron` et enregistre les requêtes « cron » des utilisateurs dans le répertoire `/var/cron/tabs` (resp. dans `/var/spool/cron` sous *GNU Linux*), sous le nom de l'utilisateur, ce qui limite à un le nombre de fichiers « cron » par utilisateur.

A l'exécution d'une commande le démon **crond** définit un environnement minimal déterminé par les variables `SHELL`, `PATH` (réduit à `/etc/bin:/sbin:/usr/bin:/usr/sbin`, sous *FreeBSD*), `HOME` et `LOGNAME`. Il est possible d'enrichir cet environnement de base en définissant des variables supplémentaires dans le fichier « cron ».

Si une commande nécessite l'entrée standard, il est nécessaire de prévoir sa redirection explicitement. Par défaut les sorties standard et d'erreurs sont envoyées vers la boîte à lettre de l'utilisateur.

Le *super*-utilisateur dispose du fichier `/etc/crontab` pour l'exécution périodique des tâches avec n'importe quel utilisateur du système.

Exemple, le fichier `/etc/crontab`.

```
# /etc/crontab - root's crontab for FreeBSD
#
# $FreeBSD: src/etc/crontab,v 1.21.2.3 2000/12/08 10:56:07 obrien Exp $
#
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
HOME=/var/log
#
#minute hour    mday    month    wday     who      command
#
*/5      *          *        *        *        root     /usr/libexec/atrun
#
# rotate log files every hour, if necessary
0        *          *        *        *        root     newsyslog
#
# do daily/weekly/monthly maintenance
1        3          *        *        *        root     periodic daily
15       4          *        *        6        root     periodic weekly
30       5          1        *        *        root     periodic monthly
#
# time zone change adjustment for wall cmos clock,
# does nothing, if you have UTC cmos clock.
```

```
# See adjkerntz(8) for details.
1,31 0-5 * * * root adjkerntz -a
```

Ce fichier au format texte, liste les processus (tâches) exécutés périodiquement. Une ligne est typiquement structurée en 7 champs (pour le fichier `/etc/crontab`, 6 pour les fichiers utilisateurs) séparés par des tabulations :

Minutes Heures Jour du mois Mois Jour de la semaine Identité Commande

Le contenu d'un champ exprime un élément de la périodicité :

- Une valeur indique à quel moment exécuter la commande. La valeur 10 dans le premier champ signifie à la 10^e minute, tandis que située dans le champ *Jour du mois*, elle signifie 10^e jour du mois ...
- Une liste de valeurs séparées par des virgules indique les moments d'exécution de la commande. La liste 2,4,6 dans le champ *mois* signifie février, avril et juin.
- Un intervalle de valeur est dénoté par la présence du tiret (-). L'intervalle 1-5 dans le champ *jour* de la semaine signifie du lundi au vendredi (bornes incluses). L'intervalle 0-23/2 dans le *champ heure* signifie (en extension) 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22 heures.
- Le méta-caractère * dénote le plus grand intervalle possible. Dans le champ *minute*, il dénote l'intervalle 0-59. La valeur */15 dans le champ *minute* signifie tous les 15 minutes (0, 15, 30, 45).

7. Exercices

Exercice 1

Lancer la commande suivante :

```
$ sleep 500 &
```

Indiquer comment obtenir le processus père de ce processus.

Terminer ce processus.

Exercice 2

Le « démon » **crond** s'exécute-t-il sur votre machine ? si oui quel est son *pid* ?

Exercice 3

Comment interdire à l'utilisateur (c.f. le login du compte personnel) d'utiliser la commande **crontab** ?

Exercice 4

Sur *GNU/Linux* lister l'arborescence complète (avec la filiation) de vos processus (*sysop*).

Sur *FreeBSD* installer le port `pstree` et faire de même.

Exercice 5

Sous votre compte (*sysop*), créez un fichier `crontab` réalisant les actions suivantes :

1. Ajout toutes les minutes du message « Hello world » suivi de la date (format AAAA-MM-JJ:hh:mm:ss) dans un fichier `/tmp/cron_`whoami`.log`.
2. Liste de vos processus toutes les 10 minutes entre 08h00 et 17h00, du lundi au vendredi.

Assurez-vous que cela fonctionne, puis supprimer-le.

Références

[1] MAN. *Manual pages*.

[2] NEMETH Evi, SNYDER Garth, et HEIN Trent H.. *Linux Administration Handbook*. Prentice Hall PTR. 2002.