

UNIX

LES PROCESSUS

Table des matières

1	Le système d'exploitation préemptif	3
2	Les processus	4
2.1	Définitions	4
2.2	Les informations liées aux processus	4
3	Gestion des processus	5
3.1	La commande ps.....	5
3.2	"pstree" pour lister les relations entre processus	5
3.3	Trier la sortie de "ps"	6
3.4	La commande top.....	6
3.5	Les commandes jobs, bg et fg	7
3.6	La commande kill.....	8

1 Le système d'exploitation préemptif

Unix est un système multi-tâches, c'est-à-dire qu'il peut exécuter plusieurs programmes à la fois.

En informatique, le multitâche préemptif désigne la capacité d'un système d'exploitation à exécuter ou arrêter une tâche planifiée en cours.

Principe de fonctionnement :

L'ordonnanceur distribue le temps du processeur entre les différents processus. Dans un système préemptif, à l'inverse d'un système collaboratif, l'ordonnanceur peut interrompre à tout moment une tâche en cours d'exécution pour permettre à une autre tâche de s'exécuter.

Dans un système d'exploitation multitâche préemptif, les processus ne sont pas autorisés à prendre un temps non défini pour s'exécuter dans le processeur. Une quantité de temps définie est attribuée à chaque processus ; si la tâche n'est pas accomplie avant la limite fixée, le processus est renvoyé dans la pile pour laisser place au processus suivant dans la file d'attente, qui est alors exécuté par le processeur. Ce droit de préemption peut tout aussi bien survenir avec des interruptions matérielles.

Certaines tâches peuvent être affectées d'une priorité ; une tâche pouvant être spécifiée comme « préemptible » ou « non préemptible ». Une tâche préemptible peut être suspendue (mise à l'état « ready ») au profit d'une tâche de priorité plus élevée ou d'une interruption. Une tâche non préemptible ne peut être suspendue qu'au profit d'une interruption. Le temps qui lui est accordé est plus long, et l'attente dans la file d'attente plus courte.

Au fur et à mesure de l'évolution des systèmes d'exploitation, les concepteurs ont quitté la logique binaire « préemptible / non préemptible » au profit de systèmes plus fins de priorités multiples. Le principe est conservé, mais les priorités des programmes sont échelonnées.

Pendant la préemption, l'état du processus (drapeaux, registres et pointeur d'instruction) est sauvé dans la mémoire. Il doit être rechargé dans le processeur pour que le code soit exécuté de nouveau : c'est la commutation de contexte.

Un système d'exploitation préemptif conserve en permanence la haute main sur les tâches exécutées par le processeur, contrairement à un système d'exploitation non préemptif, ou collaboratif, dans lequel c'est le processus en cours d'exécution qui prend la main et décide seul du moment où il la rend. L'avantage le plus évident d'un système préemptif est qu'il peut en permanence décider d'interrompre un processus, principalement si celui-ci échoue et provoque l'instabilité du système.

2 Les processus

2.1 Définitions

Un processus est une instance d'un programme en train de s'exécuter, une tâche. Le shell crée un nouveau processus pour exécuter chaque commande.

Les priorités des processus créent un phénomène appelé "ordonnancement " lié à la gestion de la mémoire.

2.2 Les informations liées aux processus

- PID : nombre identifiant un processus. Utilisé pour l'administration du processus par le noyau.
 - lors de la commande fork il y a attribution d'un PID
 - lors de la commande exec on garde le même PID.
- PPID : identité du processus père (créateur).
- UID, EUID, SUID : identifiant(s) du (des) propriétaire(s).
 - UID : UID réel (UID du créateur c'est à dire du processus père).
 - EUID : UID effectif.
 - SUID : UID sauvegarde.
- GID, EGID, SGID : identifiant(s) du groupe.
 - GID : GID réel (GID du créateur c'est à dire du processus père).
 - EGID : GID effectif.
 - SGID : GID sauvegarde.
- TTY : terminal d'attachement.
- PRI : priorité
- S : état du processus (endormi, en attente, en exécution, etc.)
- STIME : date de lancement du processus
- TIME : temps écoulé d'utilisation d'unité centrale
- SIZE : taille de la mémoire allouée.

UID, EUID, SUID sont hérités du père au moment du fork (UID=EUID=SUID). Dans un terminal, il est possible d'interagir avec le noyau ou les processus en action. Il suffit de quelques commandes pour maîtriser la gestion des tâches.

3 Gestion des processus

Les commandes qui nous intéressent sont ps, top, kill, killall, bg, fg.

Nota : Un signe \$ précède les commandes qui ne nécessitent pas des droits administrateurs, un signe # celles qui nécessitent des droits administrateurs (ces signes ne font PAS partie des commandes). Les lignes qui ne commencent pas par un signe \$ ou # correspondent au résultat de la commande précédente.

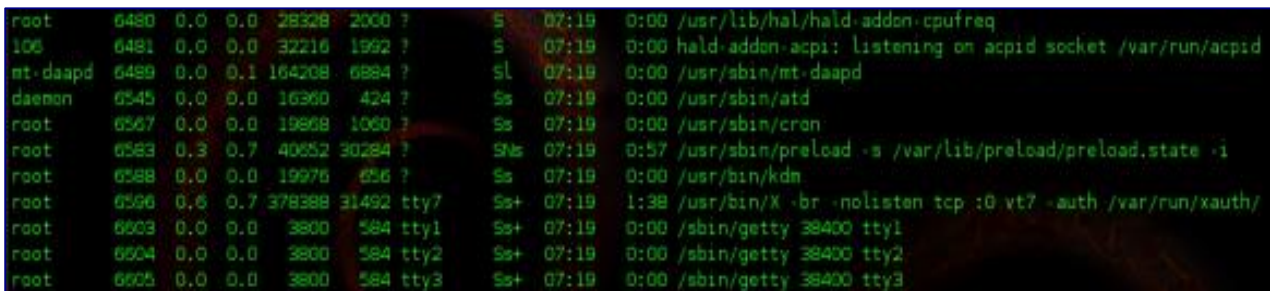
3.1 La commande ps

La commande ps permet de voir l'état des processus en cours d'exécution sur une machine. C'est une photographie de ce qui est en train de tourner sur un système.

Lancée en simple utilisateur ou en root, elle permettra de voir de nombreuses informations sur les processus. ps est rarement utilisée sans option, on utilisera rapidement des options pour avoir toutes les informations nécessaires.

Par exemple :

\$ ps -aux



user	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	TIME	COMMAND
root	6480	0.0	0.0	28328	2000	?	S	07:19	0:00 /usr/lib/hal/hald-addon-cputime
root	6481	0.0	0.0	32216	1992	?	S	07:19	0:00 hald-addon-acpi: listening on acpid socket /var/run/acpid
mt-daapd	6489	0.0	0.1	164208	6884	?	SL	07:19	0:00 /usr/sbin/mt-daapd
daemon	6545	0.0	0.0	16360	424	?	Ss	07:19	0:00 /usr/sbin/atd
root	6567	0.0	0.0	19808	1060	?	Ss	07:19	0:00 /usr/sbin/cron
root	6563	0.3	0.7	40652	30284	?	SNs	07:19	0:57 /usr/sbin/preload -s /var/lib/preload/preload.state -i
root	6588	0.0	0.0	19976	656	?	Ss	07:19	0:00 /usr/bin/kdm
root	6596	0.6	0.7	378388	31492	tty7	Ss+	07:19	1:38 /usr/bin/X -br -nolisten tcp :0 vt7 -auth /var/run/xauth/
root	6603	0.0	0.0	3800	584	tty1	Ss+	07:19	0:00 /sbin/getty 38400 tty1
root	6604	0.0	0.0	3800	584	tty2	Ss+	07:19	0:00 /sbin/getty 38400 tty2
root	6605	0.0	0.0	3800	584	tty3	Ss+	07:19	0:00 /sbin/getty 38400 tty3

On obtient donc :

- L'utilisateur,
- le PID de chaque processus : c'est son numéro d'identification,
- l'utilisation du processeur, de la mémoire vive, la date d'exécution, le temps d'exécution,
- le nom de chaque processus lancé.

On pourra également obtenir une représentation des filiations entre processus en ajoutant l'option "--forest", ou représentation en arborescence :

\$ ps -aux --forest

3.2 "pstree" pour lister les relations entre processus

Pour obtenir une représentation graphique des relations de filiation entre processus vous pouvez utiliser la commande "pstree". L'option "-p" permet d'afficher également le pid des processus qui est normalement masqué :

```
$ pstree -p
```

Les pid des processus sont listés entre parenthèses “()”, les processus fils de même nom que le processus parent entre accolades “{}”. Lorsque plusieurs occurrences d'un processus sont présentes, elles sont groupées, placées entre crochets “[]” (et accolades s'il s'agit d'un processus fils), et le nombre d'occurrences est indiqué avec une étoile “*”. Par exemple 4*[{cupsd}] pour 4 occurrences du processus fils “cupsd”.



(la capture ne montre pas les pid)

3.3 Trier la sortie de "ps"

Pour cibler un processus particuliers dans la sortie de “ps”, il est facile d'utiliser “grep” en redirigeant le résultat de “ps”:

```
$ ps -auxf | grep hald
```

```

106      6381  0.0  0.1  36892  5412 ?        Ss   07:19   0:00 /usr/sbin/hald
root     6384  0.0  0.0   15800  1232 ?        S    07:19   0:00 \_ hald-runner
root     6419  0.0  0.0   28320  2044 ?        S    07:19   0:00 \_ hald-
addon-input: Listening on /dev/input/event5 /dev/input/event3 /dev/input/event2
/dev/input/event0
root     6468  0.0  0.0   28320  2056 ?        S    07:19   0:01 \_ hald-
addon-storage: polling /dev/sr0 (every 2 sec)
root     6480  0.0  0.0   28328  2000 ?        S    07:19   0:00 \_
/usr/lib/hal/hald-addon-cpufreq
1000     26088 0.0  0.0   9340   868 pts/2    S+   16:06   0:00 \_ grep hald
  
```

On constate qu'à la dernière ligne le processus de recherche avec “grep” est lui-même listé.

3.4 La commande top

Il existe une commande qui permet d'avoir les mêmes informations que ps, mais cette fois-ci en temps réel, la commande top.

```
$ top
```

Par défaut, l'affichage se rafraîchit toutes les 3 secondes. Pour changer cela, il suffit d'utiliser l'option “delay” avec le temps en secondes, par exemple pour obtenir respectivement un rafraîchissement de 0.5s secondes :

```
$ top -d 0.5
```

Vous verrez alors que les informations sont plus importantes. On obtient, en plus de ps aux, la charge du processeur, le nombre de processus...etc.

Par ailleurs, cette fois-ci, les processus sont classés du plus gourmand en processeur au moins gourmand.

Quelques options utiles :

- `top -u utilisateur` : permet de n'avoir que les processus de "utilisateur"
- `top -p1234` : permet de n'avoir que le processus qui a le PID 1234

"top" est un programme interactif, il accepte de nombreuses options et commandes, si vous tapez la lettre "h" par exemple, vous aurez accès à l'aide qui liste les commandes disponibles :

Parmi les commandes d'usage courant, on notera

- "u" qui permet de restreindre l'affichage à un utilisateur particulier
- "k" qui permet de tuer un processus en donnant son pid (confirmation requise)
- "r" qui permet de changer la priorité d'un processus ("renice").
- "F" ou "O" qui permettent de changer la colonne qui détermine l'ordre des processus (charge cpu par défaut).
- "q" pour... quitter "top".

Nota : Il existe des versions "évoluées" ou alternatives de top : htop et atop dont le seul inconvénient par rapport à top est qu'il soit nécessaire de les installer pour les utiliser.

3.5 Les commandes jobs, bg et fg

Important : les commandes bg, fg, jobs sont des commandes propres au shell que vous avez lancé. Les exemples donnés ci-dessous ont été effectués sur bash, qui est le shell le plus utilisé. Si vous utilisez un autre shell, il faudra vous référer au manuel de celui-ci pour avoir plus de renseignements sur le contrôle des tâches.

On peut mettre une commande en "arrière-plan", c'est-à-dire que, une fois lancée, la commande ne nous bloquera pas la console jusqu'à ce qu'elle soit finie (ce qui devrait être le comportement normal) et agira sans que nous soyons obligé de nous en soucier.

Un exemple simple : je suis en console, je veux copier l'intégralité de mon disque 1 vers mon disque 2, et je sais que j'ai de multiples musiques et films libres qui prennent beaucoup de place. Le transfert sera donc très long et je ne veux pas attendre pour taper d'autres commandes en parallèle.

On utilise l'esperluette & en fin de commande pour qu'elle soit lancée en arrière-plan :

```
$ cp /toutesmesdonnees /chemin/de/destination &
```

Une fois que cela est fait, j'aimerais tout de même savoir de temps en temps où en sont mes commandes passées en arrière-plan. Il me suffit de taper la commande jobs ("travaux" en français) pour connaître toutes les commandes lancées en arrière-plan (ici, deux recherches avec la commande find).

```
$ jobs
```

```
[1]- Running      find / -iname truc &
```

```
[2]+ Running      find / -iname machin &
```

On pourra noter que chaque commande est numérotée (ici 1 et 2). C'est ce numéro d'identification qui va nous servir par la suite. Le numéro donné par job et le numéro d'un processus donné par top n'ont rien à voir. Le premier sert aux commandes lancées dans un shell, l'autre est pour le noyau. Ils n'ont donc pas la même signification, mais aussi et surtout la même référence !

Pour maintenant remettre une commande au premier plan, je peux utiliser la commande fg (pour "foreground", "premier-plan" en anglais). Là, trois solutions :

- Soit je n'ai qu'une commande en arrière-plan, et je peux utiliser fg sans argument,
- Soit j'ai plusieurs commandes en arrière-plan, et j'utilise alors le numéro donné par jobs pour savoir quelle commande je désire récupérer.

Voici donc un exemple. Si je veux revoir au premier plan la recherche de "machin" (donc le job numéro 2) :

```
$ fg %2
```

Notez bien l'utilisation de %, c'est obligatoire pour que fg trouve bien le bon processus.

C'est maintenant que les choses se corsent ... Maintenant que j'ai récupéré la main sur ma commande de recherche de "machin", je crois comprendre que je vais en avoir pour un certain temps. Je veux donc à nouveau le remettre en arrière-plan. Pour cela, je vais le stopper, pour récupérer la main sur le shell, puis lui dire de continuer en arrière-plan, grâce à la commande bg (pour "background", "arrière-plan" en anglais). Voici donc les manipulations :

1. Pour stopper le processus actuellement en premier-plan, il faut faire ctrl-z (noté aussi ^z).
2. Vérifier le numéro assigné à ce processus, à l'aide de jobs (vous pouvez remarquer qu'il est indiqué comme "stopped", c'est-à-dire arrêté) :

```
$ jobs
```

```
[1]- Running      find / -name truc &
```

```
[2]+ Stopped      find / -name machin
```

3. Puis relancez-le en arrière-plan avec bg (j'utilise le numéro 2 ici):

```
$ bg %2
```

3.6 La commande kill

La commande kill va nous servir à envoyer un signal à un processus en cours d'exécution. Cette commande est puissante, attention donc à ne pas faire n'importe quoi avec. Elle peut donc utiliser les numéros donnés par jobs et ainsi arrêter un processus. Pour arrêter ma recherche sur "machin" (voir chapitre ci-dessus), il me suffit donc d'écrire :

```
$ kill %2
```

Le processus sera alors arrêté.

"kill" peut prendre en argument le PID d'un processus, tel que renvoyé par "top" ou "pgrep". Prenons l'exemple d'une recherche du PID de "konqueror" (navigateur de fichiers/web KDE), et de son

utilisation avec “kill”:

```
$ pgrep -l konqueror
```

```
9656 konqueror
```

```
$ kill 9656
```

Par défaut “kill” envoie le signal “SIGTERM” (option “-15”), qui est reçu par le programme visé, et lui indique de se terminer proprement. Il peut arriver que ce signal soit inefficace si le programme est totalement “gelé”, dans ce cas on utilisera l'option “-9” qui envoie le signal “SIGKILL”. Ce signal n'est pas reçu par le programme, c'est le noyau qui termine immédiatement le processus, ce qui peut laisser “traîner” des processus fils orphelins, entraîner la perte du travail en cours dans le programme visé... À n'utiliser que dans les cas désespérés :

```
$ kill -9 9565
```

“kill” peut envoyer de nombreux autres signaux, ils sont détaillés dans la page de manuel de “kill”. En plus des deux précédemment cités “TERM” (option “-15”) et “KILL” (option “-9”), on peut retenir “HUP” (option -1) qui est utile pour les services (samba, sshd,...). “HUP” va entraîner le redémarrage du service visé après relecture des fichiers de configuration, utile pour prendre en compte un changement de configuration.

Pour indiquer de manière plus “conviviale” le nom du programme visé au lieu de son PID, on peut utiliser la commande “killall”:

```
$ killall konqueror
```

Attention au fait que si plusieurs processus de même nom existent, tous seront terminés.

Nota : Si le programme/processus visé par “kill” appartient à un utilisateur différent du votre, ou à “root”, vous devrez employer “kill” avec des droits “root”. Attention dans ce cas car “kill” pourra alors interrompre n'importe quel processus, et littéralement provoquer l'instabilité du système.