UNIVERSITÉ CAEN NORMANDIE

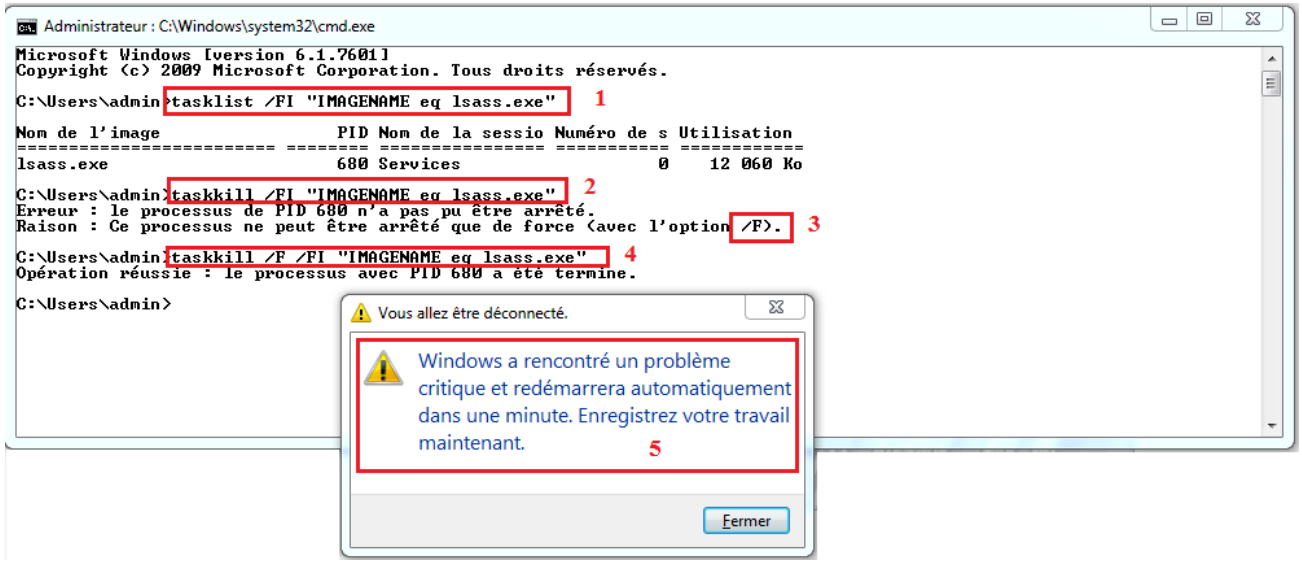
Gestion & Surveillance des Processus 15.10.2018

Un vaste programme.... et un long processus...

Le programme c'est la recette du gâteau

le processus c'est faire le gâteau et le manger, hum...☺

Auteur : Pascal Fougeray



Source : une expérience de votre prof sur son PC... juste eu le temps de sauvegarder le support de cours.....

A tester avec le processus ***System Idle Process*** et si cela fonctionne, cool la protection sous Windows

---

1 Introduction

Tout système d'exploitation sur un ordinateur quelconque est constitué d'un ensemble d'entités physiques et logiques...

Parmi les entités logiques, on trouve les processus qu'il faut savoir gérer, surtout quand l'un d'eux "plante".

Par exemple sur les derniers routeurs de chez Cisco, tournant avec des IOS XE, qui ne sont en fait que des Linux, on peut avoir des dizaines de processus.

```
PE1#show version
Cisco IOS XE Software, Version 03.14.00.S - Standard Support Release
Cisco IOS Software, CSR1000V Software (X86_64_LINUX_IOSD-UNIVERSALK9-M), Version 15.5(1)S, REL
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Thu 20-Nov-14 17:11 by mcpre
```

Toujours sur cette même plateforme, on peut voir les processus en cours avec la commande : PE1***#show processes platform*** qui renvoie

```
PE1#show processes platform
CPU utilization for five seconds: 0%, one minute: 0%, five minutes: 0%
   Pid    PPid  Status        Size  Name
-------------------------------------------------------
     1       0  S          1863680  init
     2       0  S                0  kthreadd
     3       2  S                0  migration/0
     4       2  S                0  sirq-high/0
     5       2  S                0  sirq-timer/0
     6       2  S                0  sirq-net-tx/0
     7       2  S                0  sirq-net-rx/0
     8       2  S                0  sirq-block/0
     9       2  S                0  sirq-block-iopo
    10       2  S                0  sirq-tasklet/0
    11       2  S                0  sirq-sched/0
    12       2  S                0  sirq-hrtimer/0
    13       2  S                0  sirq-rcu/0
    14       2  S                0  watchdog/0
    15       2  S                0  desched/0
    16       2  S                0  events/0
```

Oh... le beau processus init avec un PID = 1 ☺

Un autre commande telle : PE1#***show processes cpu*** renvoie

```
 534           0           2           0  0.00%  0.00%  0.00%   0 MRIB Process
 PID Runtime(ms)     Invoked      uSecs   5Sec   1Min   5Min TTY Process
 535           0           9           0  0.00%  0.00%  0.00%   0 EEM Helper Threa
 536           0         328           0  0.00%  0.00%  0.00%   0 LSD Main Proc
 538           1        1637           0  0.00%  0.00%  0.00%   0 mLDP Process
 539           0         164           0  0.00%  0.00%  0.00%   0 MFI LFD Stats Pr
 540           0         309           0  0.00%  0.00%  0.00%   0 MFI LFD Timer Pr
 541           0           2           0  0.00%  0.00%  0.00%   0 MFI LFD Main Pro
 542           0           2           0  0.00%  0.00%  0.00%   0 LSP Verification
 543           0           4           0  0.00%  0.00%  0.00%   0 MPLS IFMIB Proce
 544           0          28           0  0.00%  0.00%  0.00%   0 LDP Background
 546          21        1123          18  0.00%  0.00%  0.00%   0 LDP Hello
 547           1         166           6  0.00%  0.00%  0.00%   0 LCON Main
 PID Runtime(ms)     Invoked      uSecs   5Sec   1Min   5Min TTY Process
 548           0           2           0  0.00%  0.00%  0.00%   0 IP MPLS Service
 549           0          29           0  0.00%  0.00%  0.00%   0 IPRM
 550           2           1        2000  0.00%  0.00%  0.00%   0 LICENSE AGENT
 551           6        1611           3  0.00%  0.00%  0.00%   0 OSPF-1 Router
 552           0           9           0  0.00%  0.00%  0.00%   0 LCON Addr
 553           2         356           5  0.00%  0.00%  0.00%   0 OSPF-1 Hello

This command only shows processes inside the IOS daemon.
Please use 'show processes cpu platform'
to show processes from the underlying operating system.

PE1#show processes cpu
```

Oh... les processus ***OSPF***, ***LDP***, ***MPLS*** et même ***EEM Helper Thread***

Convaincus sur l'utilité de connaitre la gestion des processus sous Linux, Windows et Cissco, oui, alors je continue...

2 Rappels

Un processus est dynamique et correspond à l'exécution par un processeur d'un programme statique.

2.1 Processus vs Thread

Il y a les processus et les threads...

Quelques questions :

— Quelles sont les différences ?
— Pourquoi utiliser l'un et pas l'autre ?
— Peut-on faire la même chose ?

2.1.1 Processus

Un processus est une instance d'exécution d'une application. Il peut contenir plusieurs ***threads***

2.1.2 Threads

— Les threads sont des **processus légers** exécutés **dans** un processus
— Un thread est **un chemin d'exécution** au sein d'un processus.
— L'exécution des threads est **concurrente**
— Il y a au moins 1 thread par processus, nommé le thread principal
— Sa durée de vie ne peut dépasser celle du processus qui l'a créé
— Les threads d'un même processus partagent la même zone mémoire !
— Le cout de création est moindre que pour les processus lourds, car il y a échange de codes sans recopie, par échange de pointeur.

Sous Windows les threads ce sont les **dll** que Microsoft appelle modules que l'on peut voir avec la commande ***tasklist /m*** qui renvoie

```
Nom de l'image                 PID Modules
========================= ======== ============================================
System Idle Process              0 N/A
System                           4 N/A
smss.exe                       368 ntdll.dll
csrss.exe                      504 ntdll.dll, CSRSRV.dll, basesrv.DLL,
                                   winsrv.DLL, USER32.dll, GDI32.dll,
                                   kernel32.dll, KERNELBASE.dll, LPK.dll,
                                   USP10.dll, msvcrt.dll, sxssrv.DLL, sxs.dll,
                                   RPCRT4.dll, CRYPTBASE.dll, ADVAPI32.dll,
                                   sechost.dll
wininit.exe                    620 ntdll.dll, kernel32.dll, KERNELBASE.dll,
                                   USER32.dll, GDI32.dll, LPK.dll, USP10.dll,
                                   msvcrt.dll, RPCRT4.dll, sechost.dll,
                                   profapi.dll, IMM32.DLL, MSCTF.dll,
                                   nvinitx.dll, VERSION.dll, ADVAPI32.dll,
                                   RpcRtRemote.dll, apphelp.dll,
                                   CRYPTBASE.dll, WS2_32.dll, NSI.dll,
                                   mswsock.dll, wshtcpip.dll, wship6.dll,
                                   secur32.dll, SSPICLI.DLL, credssp.dll
```

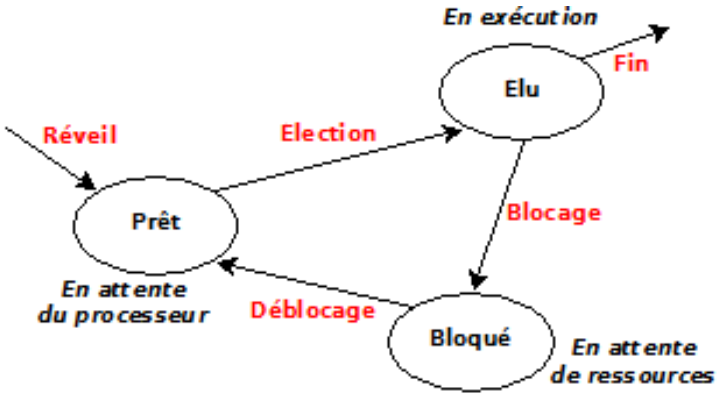
On constate la présence du PID, du nom du programme et les

2.2 États d'un processus

Sous un système Unix, la commande ***ps -l*** renvoie une ligne qui ressemble à peu près à

```
monserveur init # ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0  3296  3171  0  80   0 -  2480 wait   pts/3    00:00:00 su
4 S     0  3304  3296  0  80   0 -  2140 wait   pts/3    00:00:00 bash
4 S     0  3736  3304  0  80   0 - 53153 poll_s pts/3    00:00:02 emacs
0 R     0  3755  3304  0  80   0 -  1617 -      pts/3    00:00:00 ps
monserveur init #
```

— **Running** (R) : Le processus s'exécute sur un des processeurs (***running***) ou bien il est en attente d'exécution (***runnable***). Dans ce second cas, il est présent dans la file d'attente (***run queue***) des processus en attente d'un processeur libre.
— **Sleeping** (S) : Le processus est suspendu jusqu'à ce qu'une condition arrive, telle la réception d'un signal. Le plus souvent, c'est la fin d'une E/S qui est attendue.
— **Uninterruptible** (D) : Le processus est suspendu, mais il ne peut être interrompu par un signal appelé (***deep sleep***, souvent envoyé par la commande ***kill*** sous Linux). Généralement un processus ne peut pas être interrompu quand il accède à un périphérique dont la présence est testée (***probe***)
— **Stopped** (T) : Le processus est arrêté suite à la réception du signal SIGSTOP ou que l'application s'exécute sous le contrôle d'un déboguer (***traced***)
— **Zombie** (Z) Le processus est mort et ne consomme plus de temps CPU ni de mémoire, mais il est encore dans la table des processus. Il reste dans cet état tant que son père n'a pas pris connaissance de sa fin.



Il en va de même sous windows, sauf qu'il n'y a que 3 états, obtenus avec la commande ***tasklist /v***

— **Running** tout tourne ☺
— **Not Responding**, pas de réponse, indique un processus qui devrait être arrêté.
— **Unknown** ce statut *Unknown*, inconnu, peut faire référence à un processus normal, mais "Not Responding" (pas de réponse) indique un processus qui devrait être arrêté.

2.3 Ordonnancement

Le souci : **À un instant t, il y a souvent davantage de processus à exécuter que de processeurs.**

Un des rôles de l'OS est de permettre à tous ses processus d'être exécuté à un moment ou un autre et d'utiliser au mieux le processeur, c'est le rôle de **l'ordonnanceur** du noyau, on parle aussi de ***Scheduler***. Pour que chaque tâche s'exécute sans se préoccuper des autres et/ou aussi pour exécuter les tâches selon les contraintes imposées au système, par exemple les contraintes temporelles dans le cas d'un **OS temps réel**, l'ordonnanceur du noyau du système effectue des **commutations de contexte**. C'est à dire, il passe d'un processus à un autre.

Cette commutation de contexte prend du temps, on peut donc en conclure qu'un système multi-processus est plus lent qu'un système mono-processus tout e permettant de gagner du temps... !

Une explication ?

En groupe on travaille plus vite, c'est vrai mais est-ce que 8 personnes font le travail de 8 personnes seules ?

Et bien non, il faut synchroniser le tout et les plus lents ralentissent les plus rapides et ilfaut sauvegarder le contexte.

2.4 Contexte d'un processus et son SAV

Le contexte d'un processus, c'est son état et les données sur lesquelles il travaille à un temps t.

Une commutation de contexte (*context switch*) en informatique consiste à sauvegarder l'état d'un processus pour restaurer à la place celui d'un autre dans le cadre de l'ordonnancement d'un OS multitâche.

Elle peut être plus ou moins couteuse en temps processeur suivant l'architecture matérielle, l'OS, ou le type de processus.

En effet, dans le cas des processus lourds, donc pas les ***theads,*** cela nécessite toujours un **changement d'espace d'adressage**, alors que les processus légers,les ***threads***, de même père partagent ce dernier, qui n'a alors pas besoin d'être rechargé.

Le contexte sauvegardé doit au minimum inclure une portion notable de l'état du processeur (registres généraux, registres d'états, etc.) ainsi que, pour certains systèmes, les données nécessaires au système d'exploitation pour gérer ce processus.

La commutation de contexte invoque au moins trois étapes. Par exemple, en présumant que l'on veut commuter l'utilisation du processeur par le processus P1 vers le processus P2 :

Sauvegarder le contexte du processus P1 quelque part en mémoire (usuellement sur la pile de P1).

Retrouver le contexte de P2 en mémoire (usuellement sur la pile de P2).

Restaurer le contexte de P2 dans le processeur.

Reprendre l'exécution de P2 à son point de dernière exécution.

Certains processeurs peuvent sauvegarder et restaurer le contexte du processus en interne, évitant ainsi d'avoir à sauvegarder ce contexte en mémoire vive.

2.5 Processus fils : fork

Allez un peu de langage C,juste pour le fun et montrer comment cela se passe sous Linux, mais aussi avec les autres SE !

Un nouveau processus peut-être créé en appelant la primitive ***fork***.

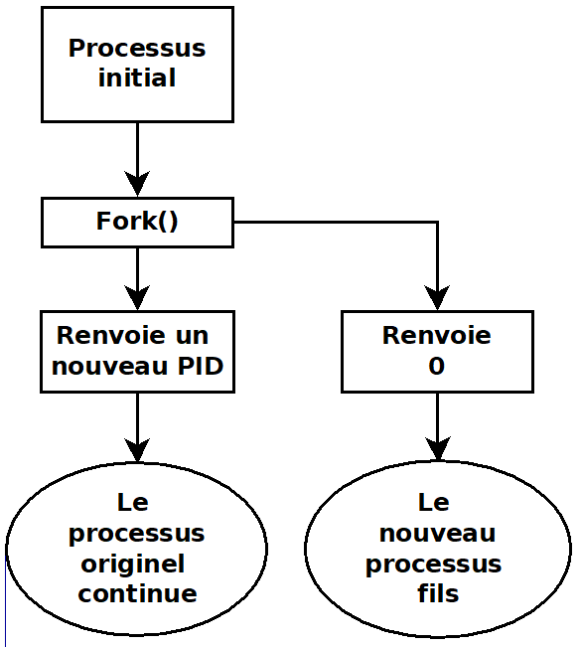
#include <unistd.h>

pid_t fork(void)

Cet appel système copie le processus actif en insérant dans la table des processus une nouvelle entrée dotée des mêmes attributs, ce nouveau processus est quasiment identique à l'original, il exécute le même code mais avec ses propres espaces de données, environnement et descripteurs de fichiers.

L'appel système ***fork*** dans le père renvoie le **PID** du processus fils. Celui-ci continue de s'exécuter comme l'original, à la différence que ***fork*** renvoie 0 dans le processus fils.

Cela permet de savoir qui est qui.




```
1  #include <sys/types.h>
   #include <unistd.h>
   #include <stdio.h>

   int main(){
6    pid_t pid;
     char *message;
     int n;
     printf("Lancement du fork\n");
     pid=fork();
11   switch(pid) {
       case -1:
         perror("echec de fork");
         return 1;
       case 0:
16       message = "le fils";
         n=10;
         break;
       default:
         message = "le papa";
21       n=3;
         break;
       }
     for(; n>0; n--){
       printf("%s, pid : %i et papa : %i \n",\
26        message,getpid(),getppid());
       sleep(1);
     }
     return 0;
   }
```

L'utilisation typique de l'appel **fork** est la suivante :

1. ***pid_t pid = fork();***
2. ***if pid (pid==0) {***
3. ***/* Instructions du fils */ }***
4. ***else if (pid >0) { // pid du fils <>0***
5. ***/* Instructions du père */ }***
6. ***else {***
7. ***/* Erreur (Instruction du père) */ }***

Le primitive ***fork*** renvoie -1 en cas d'échec souvent en raison du nombre limité de processus fils qu'un père peut posséder (**CHILD_MAX**), un manque d'espace empêchant la création d'une entrée dans la table des processus ou une mémoire virtuelle insuffisante.

Dans l'exemple précédent, le programme s'exécute comme 2 processus, le père affiche 3 messages et le fils 8. Ce programme ne fonctionne pas correctement, pourquoi ?

Réponse : Il faudrait synchroniser les processus, hors programme...

2.6 Les *daemons*

Un ***daemons*** est un processus en tâche de fond. Il rend un service quand on lui demande. Par exemple ***httpd***.

3 La mise en pratique

Voir l'état des processus etc...sous divers OS ....
Et comme tout est un ordinateur... il y a du monde !

3.1 Sous Linux

Sous Linux, le **noyau** lance, gère les processus et contrôle leurs échanges avec les E/S.
Le premier processus, ancêtre de tous les autres, est ***init***
Tiens en parlant d'ancêtre... **INIT** est l'ancêtre... car depuis quelques années, ce n'est plus ***Init*** qui démarre Linux et lance les autres processus mais ***kthreadd***
Le système actuel et qui remplace tout est ***Systemd***
Un excellent article :
http://linuxfr.org/news/systemd-l-init-martyrise-l-init-bafoue-mais-l-init-libere
Tous les autres sont créés par un processus parent et appartiennent à un utilisateur.
Chaque processus est identifié par un numéro unique, son PID.
Pour lancer un processus en tâche de fond, faire suivre la commande de ***l'esperluette &***, ce qui ordonne au processus parent de "reprendre la main", sans attendre la fin du processus "fils".

3.1.1 La commande ps et autres

La commande ***ps*** pour ***Processus Status*** permet de voir les processus qui sont lancés.
Plus d'informations avec ***man ps*** ☺
— **ps aux** : affiche la liste de tous les processus, avec leur numéro PID, et d'autres informations

USER à quel utilisateur appartient le processus.

PID est le numéro qui identifie le processus

%CPU en % les ressources du microprocesseur utilisées par le processus.

%MEM en % les ressources en mémoire vive utilisées par le processus.

RSS mémoire réellement utilisée en ko par le processus.

START l'heure à laquelle le processus a été lancé.

La commande **pidof** : Elle permet de connaitre la liste des PID des processus d'un programme, exemple **pidof apache2**
La commande **pstree** : affiche la filiation des processus sous forme arborescente.
Les commandes **kill** et **killall** : On peut gérer les processus en leur envoyant des signaux par l'intermédiaire de ces commandes, suivant que l'on connaisse le PID du processus, ou bien son nom.
Quelques exemples d'utilisations :

3.1.2 Init et les [XXXXXd]

Sous Linux le premier processus lancé par le ***kernel*** est ***init***, comme le montre un extrait des sources du ***kernel***, le fichier ***main.c*** se trouvant dans le répertoire ***/usr/src/linux-source.***/init/main.c***

```
if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
                return 0;
        pr_err("Failed to execute %s (error %d).  Attempting defaults...\n",
                execute_command, ret);
}
if (!try_to_run_init_process("/sbin/init") ||
    !try_to_run_init_process("/etc/init") ||
    !try_to_run_init_process("/bin/init") ||
    !try_to_run_init_process("/bin/sh"))
        return 0;

panic("No working init found.  Try passing init= option to kernel. "
      "See Linux Documentation/init.txt for guidance.");
}
```

Si pas de programme init alors juste un shell et on se débrouille !!!

Si on lance la commande ***ps***

3.1.3 /var/run /

Ce répertoire contient des fichiers d'information système décrivant le système depuis son démarrage.
Les fichiers de ce répertoire sont supprimés au début du processus de démarrage.
Les fichiers d'identification de processus (PID), sont placés dans ce répertoire.
Ils sont sous la forme ***nom_programme.pid***.
Exemple, le fichier PID de snmpd est nommé ***/var/run/snmpd.pid***.

3.1.4 Envoyer un signal

Un signal est un **message asynchrone** envoyé à un processus pour lui indiquer une information.
Les signaux reçus par un processus ont plusieurs origines possibles :

- L'utilisateur, au clavier, le signal **sigint** à tous les processus lancés depuis ce terminal (qui n'ont pas été mis en arrière plan) quand l'utilisateur tape le caractère d'interruption ctrl-C. De même, il envoie ***sigquit*** quand l'utilisateur tape ctrl-\1.
- Le ***kernel***, il envoie ***sighup*** lorsque la connexion avec le terminal est fermée, ou bien quand l'utilisateur s'est déconnecté.
- L'utilisateur, par les commandes ***kill*** et ***killall***. Cette commande permet d'envoyer un signal quelconque à un processus quelconque.
  Exemple, ***kill -KILL 1664*** envoie le signal ***sigkill*** au processus de PID 1664, ce qui a pour effet de terminer ce processus puisqu'on le tue...
- D'**un autre processus**, via le ***kernel***, qui exécute un appel système ***kill***, le cas précédent en est un exemple particulier.
- Le ***kernel***, pour des processus qui se comportent mal.
  - Un processus qui effectue une division par zéro reçoit un signal ***sigfpe***,
  - Un processus qui viole l'espace mémoire d'un autre processus reçoit le signal ***sigsegv***
- Le fils, via le ***kernel***, pour prévenir un processus père que son environnement a changé.
  - Quand un processus se termine, son père reçoit le signal ***sigchld***.

3.1.5 cron & anacron

2 processus daemon qui permettent de lancer des tâches à des instants précis et réguliers, par exemple tous les jours, toutes les semaines, tous les mois etc...

cron

utilisé sur les serveurs allumés 24/24 et 7/7...

C'est un démon qui s'appuie sur le fichier de configuration nommé ***/etc/crontab*** pour déterminer quels programmes et / ou scripts à lancer à quelle date et heure.

Ce fichier contient 6 champs, 5 pour la date et l'heure et le sixième pour ce qui est à faire.

minute (0-59) heure (0-23), jour du mois (1-31), mois de l'année (1-12), jour de la semaine (0-6 avec 0=Dimanche) #

Exemples :

0 5 * * * /home/pascal/bin/sav-cours.sh

Lance le script sav-cours.sh tous les jours à 5h du matin[1]

0 */3 * * * /home/user/bin/downloadSerie.sh

anacron

C'est l'équivalent du **cron** mais pour les machines qui ne restent pas allumées 24/24 7/7, donc les portables par exemple

Son fonctionnement est contrôlé par le fichier ***/etc/anacrontab***.

Ce fichier contient 4 champs

Période	Délai	Le travail à faire	La commande correspondante
1 ou 7 ou 30	XX		
jour Semaine Mois	Nb minutes après allumage		
1	30	travail.daily	/bin/bash /home/pascal/sav-cours.sh

Dans ce cas, le fichier travail.daily contiendra la date du jour et la commande /bin/bash /home/pascal/sav-cours.sh sera lancée 30 minutes après le démarrage du portable

Plus d'informations : **man** et http://voidandany.free.fr/index.php/planificateurs-de-taches-cron-et-anacron-et-leurs-interactions/

1. C'est un exemple... je dors encore à cette heure, sauf pour partir en vacances fin d'éviter tous ceux qui partent le même jour
   Bon je suis prof et je peux partir un mardi moi ☺

3.2 Sous Windows

Un vaste programme.... et un long processus...

3.2.1 La commande *tasklist*

C'est un peu l'équivalent de la commande ***ps*** sous Linux
Pour plus d'informations :
http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/tasklist.mspx?mfr=true

3.2.2 Application, Processus & Service

Dans la philosophie Windows,

— une application est un programme avec une interface graphique tels *Firefox*, *Lyx*, *Word* etc..

— un processus est l'image d'un programme exécutable, donc un fichier dont l'extension est **exe** ou ***msc***. Ils se situent généralement dans le répertoire ***\Windows\System32***
Il a un propriétaire

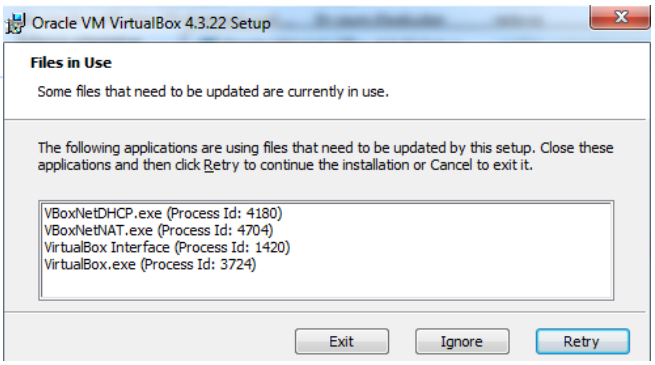
— Un service est un processus en tâche de fond, généralement lancé par le programme **svchost.exe** (voir plus loin)
Il n'a pas de propriétaire réel, mais un utilisateur virtuel appelé ***SYSTEM***. On peut le voir en allant dans la liste de processus après un CTRL+ALT+SUPPR.

Exemple : Quand on lance l'application ***Virtualbox***, et une machine virtuelle, on n'a qu'une seule application de lancée, alors que l'on a quelques processus, comme le montre les figures suivantes.

La première montre les processus lancés par cette application

La seconde montre les processus qui bloquent si on essaye de faire une mise à jour de l'application ***virtualbox***. Ici de 4.3.20 à 4.3.22

VBoxNetDHCP.exe	admin	00	1 700 K	VBoxNetDHCP.exe
VBoxNetDHCP.exe	admin	00	1 308 K	VBoxNetDHCP.exe
VBoxNetDHCP.exe	admin	00	9 968 K	VBoxNetDHCP.exe
VBoxNetNAT.exe	admin	01	18 648 K	VBoxNetNAT.exe
VBoxNetNAT.exe	admin	00	1 704 K	VBoxNetNAT.exe
VBoxNetNAT.exe	admin	00	1 304 K	VBoxNetNAT.exe
VBoxSVC.exe	admin	00	11 004 K	VirtualBox Interface
VirtualBox.exe	admin	00	1 308 K	VirtualBox.exe
VirtualBox.exe	admin	06	94 492 K	VirtualBox.exe
VirtualBox.exe	admin	00	1 704 K	VirtualBox.exe
VirtualBox.exe	admin	00	26 380 K	VirtualBox.exe



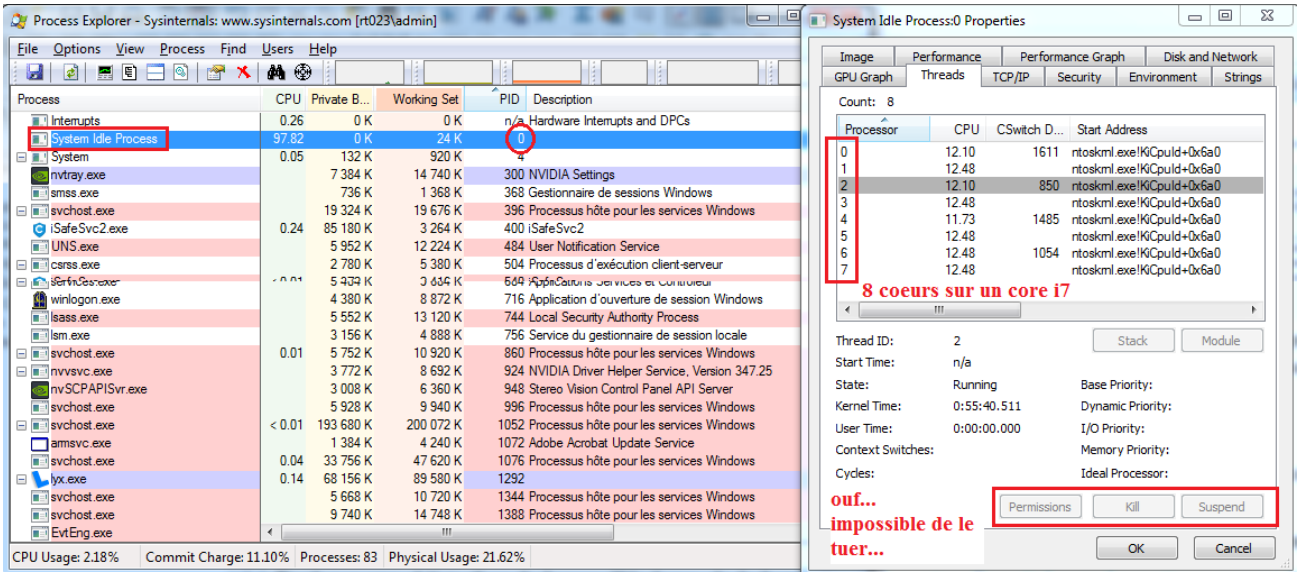
Sous Windows la gestion des processus se fait de manière plus graphique, un bon clic et on stoppe ou démarre un processus.
On ne gère pas de la même façon les processus dits "courants" et les processus dits de "service" tels les ***daemons*** sous Linux.

3.2.3 Les principaux processus windows

System-Idle-Process Le premier processus, dont le PID est 0, il gère les processeurs ou plutôt les cœurs. Il est constitué d'autant de ***threads*** que le processeur a de cœurs.

Les threads sont nommés : ***ntoskrnl.exe*** ☺

Dans mon cas, j'ai un core i7 donc 8 cœurs !



Csrss.exe ***Client Server Run-time Subsystem***. Essentiel il doit fonctionner en permanence. Il gère les applications consoles, la création et la destruction de ***threads.***

Explorer.exe L'interface du bureau Windows, de la barre des tâches, etc... Pas vital pour le système, on peut l'arrêter pour le relancer via le gestionnaire des tâches. Mais avec le risque de n'avoir plus que le fond d'écran...

Mtask.exe Service de planification de tâches, responsable du lancement des tâches à un instant précis que l'on a choisi, équivalent de ***cron*** sous **linux**

Services.exe Gestionnaire de contrôle des services, ***Service Control Manager***. Responsable du démarrage et de l'arrêt ainsi que de l'interaction avec les services système.

Smss.exe Il s'agit du sous-système de gestion de session (***Session Manager Subsystem***). Responsable du démarrage de la session utilisateur.

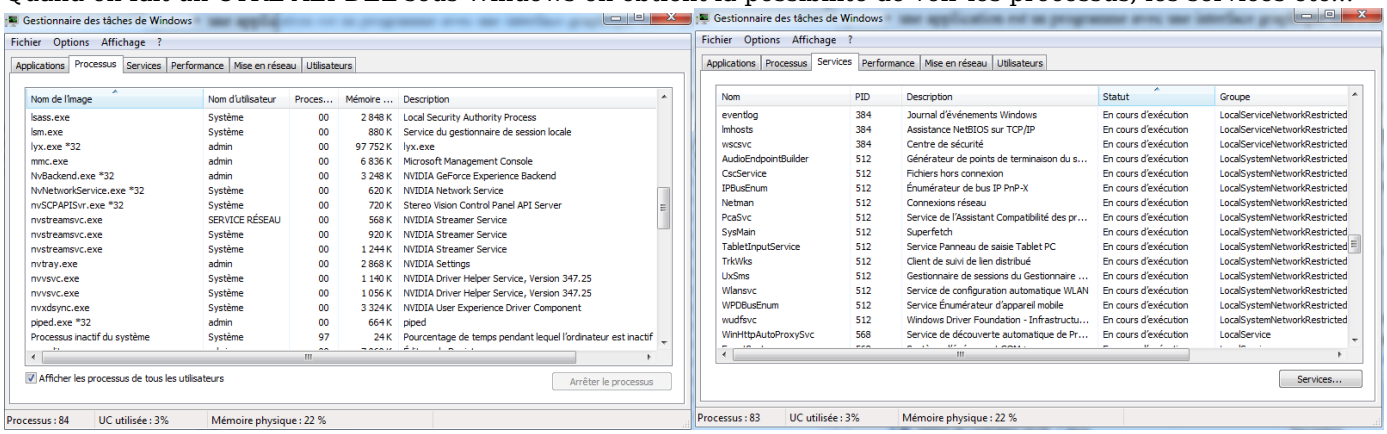
Ce processus est responsable entre autres du lancement des processus ***Winlogon*** et ***Win32*** (***csrss.exe***) et du positionnement des variables système. Après qu'il ait lancé ces processus, il attend que Winlogon ou Csrss se termine.

Spoolsv.exe Responsable de la gestion des travaux d'impression.

Svchost.exe Processus générique, il fonctionne en tant qu'hôte pour d'autres processus tournant à partir de Dlls, il peut y avoir plusieurs entrées pour ce processus (voir plus loin)

System La plupart des ***threads*** du mode noyau, il en tant que processus System.

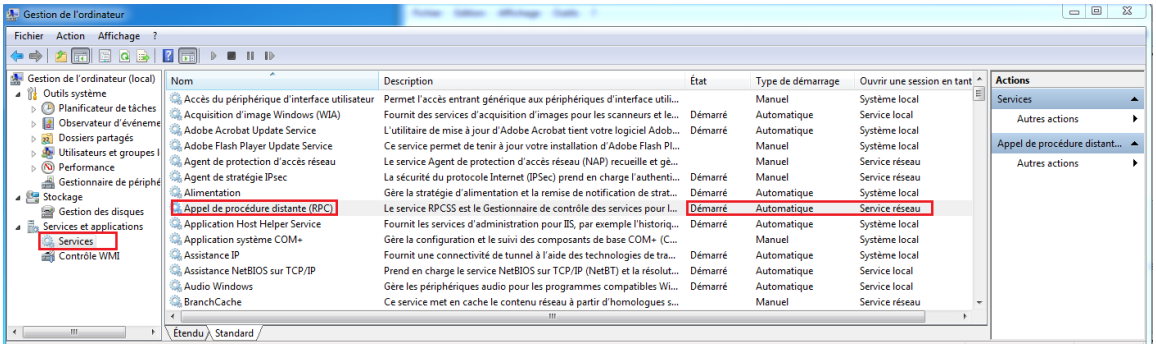
Quand on fait un CTRL-ALT-DEL sous Windows on obtient la possibilité de voir les processus, les services etc...



3.2.4 Les services ou daemons

Par exemple si on désire stopper ou démarrer le service SNMP, il faudra faire :

Démarrer, **Ordinateur** (bouton droit) **gérer** , ou bien lancer le programme ***services.msc*** et on obtient la fenêtre suivante :



Ici par exemple nous avons le processus service : RPC pour Remote Procedure Call ou Appel de Procédure Distante qui gère COM et DCOM sous Windows.

C'est un service Réseau qui est démarré automatiquement au lancement du système d'exploitation.

Un conseil n'arrêtez pas ce service...

3.2.4.1 Svchost.exe C'est un processus générique, ***generic host process*** pour les services exécutés à partir de bibliothèques dynamiques, les DLL. Elles contiennent le code du service à lancer mais on ne peut pas les lancer, alors c'est le ce processus qui s'en charge.

Par exemple pour le service client DHCP, c'est la DLL ***\Windows\System32\dhcp***

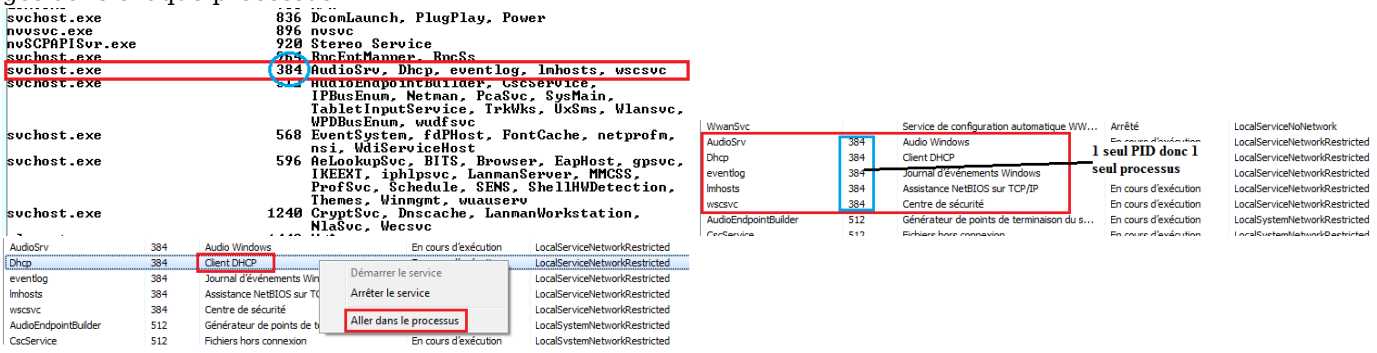
Pour fonctionner, Windows nécessite de nombreux services : tels l'interface, le pare feu, les connexions réseau, etc...

Pour éviter qu'un seul service qui plante entraine l'arrêt des autres services, ils sont regroupés par catégories dans des instances distinctes du processus **svchost.exe**.

Un processus **svchost.exe** peut alors contenir les services qui gèrent le parefeu et/ou ceux qui gèrent l'interface utilisateur etc....

C'est pour cela qu'il ne faut pas tuer ces processus **Svchost.exe**, mais plutôt utiliser l'interface des services pour stopper tel ou tel service.

L'image suivante est obtenue à l'aide de la commande ***tasklist /SVC*** permettant ainsi d'afficher les services hébergés dans chaque processus



3.2.5 Lister & Tuer un ou des processus

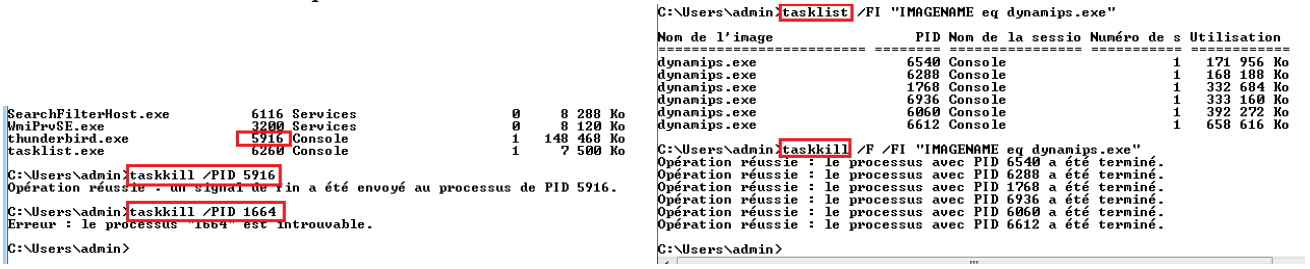
Comme nous venons de le voir, il ne faut pas tuer n'importe quel processus sous Windows ! ! !

Il y a 2 méthodes,

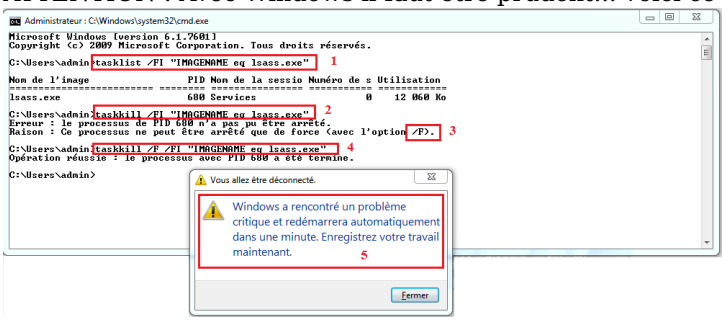
1. la graphique très pratique sauf si on a plusieurs processus de même nom à tuer ou à lister
2. en CLI, avec les commandes ***tasklist*** et ***taskkill***

La commande sous Windows est : ***taskkill*** dont la définition est : ***Cet outil est utilisé pour arrêter des tâches par id de processus (PID) ou nom d'image.***

taskkill /? donne les possibilités.



ATTENTION : Avec Windows il faut être prudent... voici ce que peut donner une simple commande ...

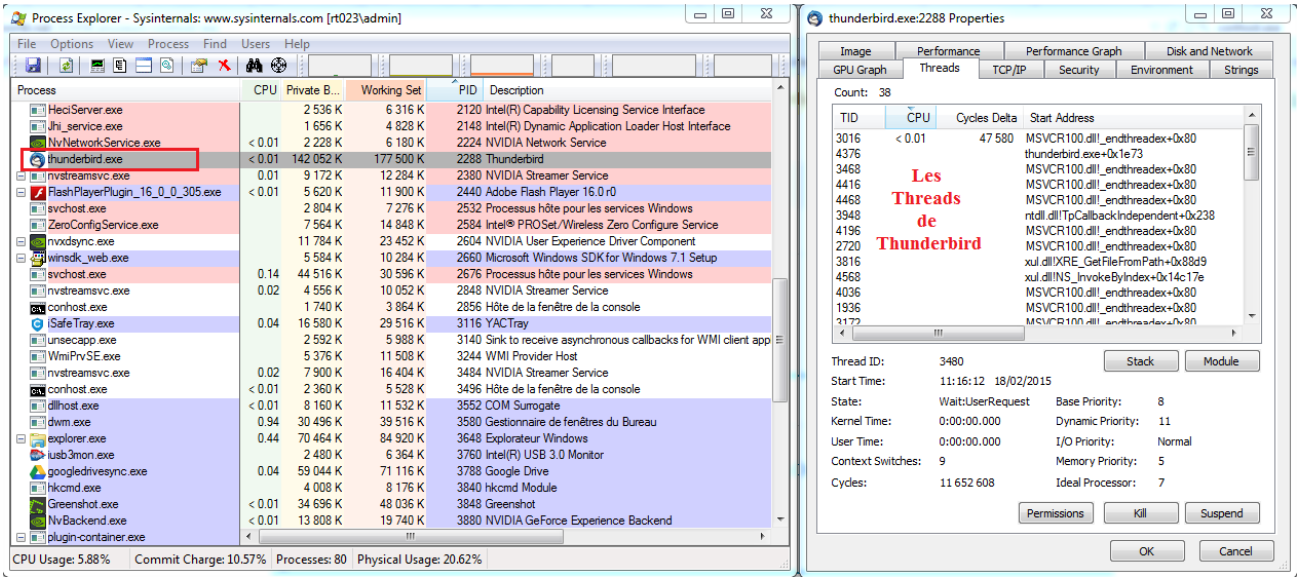


Et oui... il est possible planter Windows et de l'obliger à démarrer en étant un simple utilisateur qui tue un processus système...

3.2.6 *Process Explorer*

Si vous voulez gérer avec minutie les processus sous Windows, je vous conseille l'utilitaire ***Process Explorer*** que vous pouvez trouver à l'url : https://technet.microsoft.com/en-us/sysinternals/bb896653

C'est juste un exécutable que vous lacez en ligne de commande : ***procexp.exe*** et vous obtenez :



3.3 Chez Cisco

VOIR : http://blog.section9.be/post/2008/02/24/CISCO-IOS-%28Part-5%29 ! ! !

Comme nous l'avons vu en introduction, il est possible de gérer les processus tournant sur un routeur, un switch ou tout élément actif de ce fabricant.

Par exemple la commande : R3***#show process | i CPU|OSPF*** renvoie les processus relatifs à ***OSPF*** et ***LDP*** comme le montre la figure suivante

```
R3#show process | i CPU|OSPF
CPU utilization for five seconds: 0%/0%; one minute: 1%; five minutes: 0%
   3 ME  60C34E08           0        347       0 7904/9000   0 OSPF Hello
 109 ME  60C34E08           0          1       0 8752/9000   0 OSPF Hello
 145 ME  60C34650           8         42     190 8188/9000   0 OSPF Router
 159 ME  60C34650          24       3097       7 7600/9000   0 OSPF Router
R3#show process | i CPU|LDP
CPU utilization for five seconds: 0%/0%; one minute: 0%; five minutes: 0%
  85 Mwe 6141F608           4          5     800 5744/6000   0 AToM LDP manager
 151 Mwe 612E35D0          28         55     509 5592/6000   0 LDP
R3#
```

Pour tuer un processus, c'est la commande ? ? ?

Tout dépend de l'IOS que vous allez utiliser.

Mais vous devez être conscient qu'un IOS est un OS comme un autre. Jusqu'à maintenant vous avez manipulé sur des "vieux" systèmes, les ios des ISR ***Integrated Services Router***, ce sont les routeurs de bas de gamme et qui servent pour les petits systèmes.

Mais avec des routeurs ou des switchs tels les

— ASR 9000 Series Aggregation Services Routers

— Cisco XR 12000/12000 Series Router
— Cisco Carrier Routing System

Ayant des ios tels XE, XR, NX etc... fonctionnant avec plus de 256 processeurs, ayant des débits supérieurs à 1000 Tbps, ayant plusieurs centaines de processus tournant en parallèle et pouvant être configurés par plusieurs personnes à la fois... donc des ordinateurs mufti-tâches et multi-utilisateurs ! ! !, il faut avoir compris le fonctionnement d'un système informatique, aussi bien la partie hardware que la partie software.

4 Conclusion

Une connaissance fine du fonctionnement des processus est fondamentale pour le dépannage des applications, des systèmes logiques que vous allez devoir gérer, aussi bien en tant qu'administrateur système que administrateur réseau. ☺

L'avenir c'est que tout est ordinateur, tout est fichier, tout est objet (python), tout est virtuel... et que c'est mon dernier CM de l'année ☺

Dans les Data Center on parle de : 1 **VM** <==> 1 **process** <==> 1 **IP** <==> 1 **dialogue** !