Processus

1. Notion de processus

Dans les années 1970 les ordinateurs personnels n'étaient pas capables d'exécuter plusieurs tâches à la fois : on lançait un programme et on y restait jusqu'à ce que celui-ci plante ou se termine. Les systèmes d'exploitation récents (Windows, Linux ou osX par exemple) permettent d'exécuter plusieurs tâches simultanément - ou en tous cas, donner l'impression que celles-ci s'exécutent en même temps. A un instant donné, il n'y a donc pas un mais plusieurs programmes qui sont en cours d'exécution sur un ordinateur : on les nomme **processus**. Une des tâches du système d'exploitation est d'allouer à chacun des processus les ressources dont il a besoin en termes de mémoire, entrées-sorties ou temps processeur, et de s'assurer que les processus ne se gênent pas les uns les autres.

Un système d'exploitation gère l'ensemble des ressources et sa fonction peut se résumer en :

- la gestion des processus ;
- la gestion des fichiers ;
- la gestion de la mémoire ;
- la gestion des périphériques ;
- le traitement des entrées-sorties ;
- la sécurisation du système.

Quand on lance l'exécution d'un programme à partir d'un OS (Operating System : système d'exploitation) en cliquant par exemple sur une icône, cela déclenche la création d'un processus pour exécuter le programme. D'autres processus peuvent être déclenchés à partir d'un processus (on dira que ce sont des processus fils).
Certains fonctionnent en permanence en arrière-plan, comme un antivirus et ceux qui s'exécutent dès le démarrage de la machine (les services sous Windows ou les daemons sous Linux).
On peut dire qu'il y a deux grandes catégories de processus :

- ceux créés par un utilisateur ;
- ceux créés par le système.

Il ne faut pas confondre le fichier contenant un programme (portent souvent l'extension exe sous Windows) et le ou les processus qu'ils engendrent quand ils sont exécutés. Un programme est juste un fichier contenant une suite d'instructions (firefox.exe par exemple) alors que les processus sont des instances de ce programme ainsi que les ressources nécessaires à leur exécution (plusieurs fenêtres de firefox ouvertes en même temps).

2. Création d'un processus

La création d'un processus peut intervenir :

- au démarrage du système ;
- par un appel d'un autre processus ;
- par une action d'un utilisateur (lancement d'application).

Sous Linux, la création d'un processus se fait par clonage d'un autre processus au travers d'un appel système : **`fork()`**.

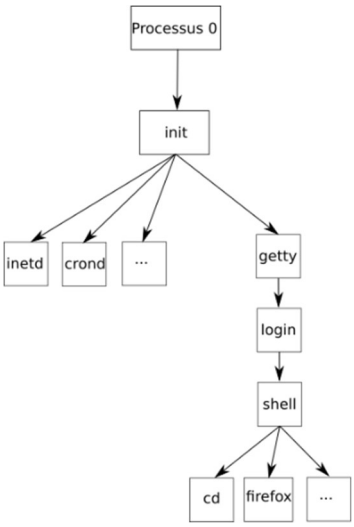
- Le processus qui fait appel à **`fork()`** est appelé processus père.
- Le processus qui est ainsi créé par clonage est le processus fils.
- Après le clonage, un processus peut remplacer son programme par un autre programme grâce à l'appel système **`exec()`**.

3. PID – PPID

Un processus est caractérisé par un identifiant unique : son **PID** (Process Identifier). Lorsqu'un processus engendre un fils, l'OS génère un nouveau numéro de processus pour le fils. Le fils connaît aussi le numéro de son père : le **PPID** (Parent Process Identifier).

4. Arborescence de processus

Un processus père engendre un ou plusieurs fils qui à leur tour engendrent des fils...
Sous Linux, **le tout premier processus appelé processus 0** ou encore **swapper** est créé à partir de "rien" (il n'est le fils d'aucun processus).
Ensuite, ce **processus 0** crée un **processus** souvent appelé **init** (init est donc le fils du processus 0) ; init a donc un PID de 1 et un PPID de 0.
À partir de init, les processus nécessaires au bon fonctionnement du système sont créés (par exemple les processus inetd, crond, …, getty). Puis d'autres processus sont créés à partir des fils de init...



Sous Linux, la commande **`pstree`** permet d'afficher l'arborescence des processus en cours.
✍ Dans le terminal du Raspberry Pi, entrer la commande **`pstree`** et observer l'arbre obtenu.
✍ Lancer le logiciel **Thonny**, entrer à nouveau la commande **`pstree`** et observer à nouveau l'arbre. Comparer les deux arbres.

5. Observer les processus

5.1. Vue statique des processus

La commande **`ps -ef`** permet d'afficher les processus au moment de la validation de la commande.

```
UID          PID  PPID  C STIME TTY          TIME CMD
pi          1680   559  0 10:45 ?        00:00:00 /usr/lib/dconf/dconf-service
root        1687     2  0 10:45 ?        00:00:00 [kworker/3:2-events_power_efficient]
root        1688     2  0 10:46 ?        00:00:00 [kworker/1:2-events]
pi          1689   698 33 10:46 ?        00:00:03 /usr/bin/python3 /usr/bin/thonny
pi          1694  1689  8 10:46 ?        00:00:00 /usr/bin/python3 -u -B -m thonny.plugins.cpython /home/pi
root        1697     2  0 10:46 ?        00:00:00 [kworker/2:2-events]
pi          1701   870  0 10:46 pts/0    00:00:00 ps -ef
```

- UID : identifiant de l'utilisateur qui a lancé la commande ;
- PID : identifiant du processus ;
- PPID : PID du père du processus ;
- C : partie entière du pourcentage d'utilisation du processeur par rapport au temps de vie des processus ;
- STIME : heure de lancement du processus ;
- TTY : nom du terminal de contrôle ;
- TIME : temps d'exécution ;
- CMD : nom de la commande du processus ;

Dans l'exemple ci-dessus, l'identifiant du processus à l'origine de tous les processus concernant le logiciel Thonny est 1689.
Le processus parent (PPID) à l'origine de tous les processus concernant le logiciel Thonny est 698.

✍ Ouvrir le terminal et tapez la commande suivante : **`ps -ef`** afin d'observer les processus en cours.
✍ Quel est le PID du processus de la commande **`ps-ef`** ?

✍ Ouvrir le navigateur Web Chromium et taper à nouveau la commande **`ps -ef`**.
✍ Identifier le PID à l'origine de tous les processus concernant le navigateur.
✍ Identifier le PPID à l'origine de tous les processus concernant le navigateur.
✍ Concernant le navigateur, quel est l'identifiant du processus dont le temps d'exécution est le plus long ?

5.2. Vue dynamique

La commande top permet d'afficher une vue dynamique des processus en cours.

```
top - 11:40:07 up  1:44,  2 users,  load average: 0,20, 0,44, 0,51
Tasks: 150 total,   1 running, 149 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,5 us,  1,3 sy,  0,0 ni, 96,1 id,  0,1 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :    924,2 total,    142,8 free,    308,2 used,    473,3 buff/cache
MiB Swap:    100,0 total,    100,0 free,      0,0 used.    511,2 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1689 pi        20   0   80836  32820  13372 S   5,9   3,5   3:14.70 thonny
  509 root      20   0  278704  91116  44296 S   2,0   9,6   1:24.35 Xorg
  856 pi        20   0  238772  51924  38836 S   2,0   5,5   0:49.93 lxterminal
 1694 pi        20   0   34972  14952   7656 S   2,0   1,6   1:02.55 python3
 2047 root      20   0       0      0      0 I   1,0   0,0   0:02.63 kworker/2:0-events
 2053 pi        20   0   10444   3028   2468 R   1,0   0,3   0:06.12 top
  371 nobody    20   0    4320   2152   1980 S   0,3   0,2   0:05.54 thd
 1863 pi        20   0  386432 108924  76624 S   0,3  11,5   0:19.94 chromium-browse
    1 root      20   0   33772   8108   6424 S   0,0   0,9   0:05.94 systemd
    2 root      20   0       0      0      0 S   0,0   0,0   0:00.02 kthreadd
    3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par_gp
```

✍ Dans le terminal, taper la commande **`top`**.
✍ Ouvrir le logiciel Thonny et identifier son PID.

La commande **`kill`** permet de fermer un processus.
✍ Ouvrir un deuxième terminal et taper le commande **`kill`** suivi du PID du processus Thonny (exemple : **`kill 1689`**) et vérifier la fermeture du logiciel Thonny.
✍ Donner la commande permettant de fermer le terminal.
✍ Vérifier votre commande dans le terminal.

6. Initialisation du noyau et premiers processus

Le **bootloader** (chargeur d'amorçage) est un petit programme, inscrit sur le disque dur, qui est lu et exécuté par le **BIOS** ou l'**UEFI** et qui a la responsabilité de charger le noyau en mémoire et de lancer son exécution.

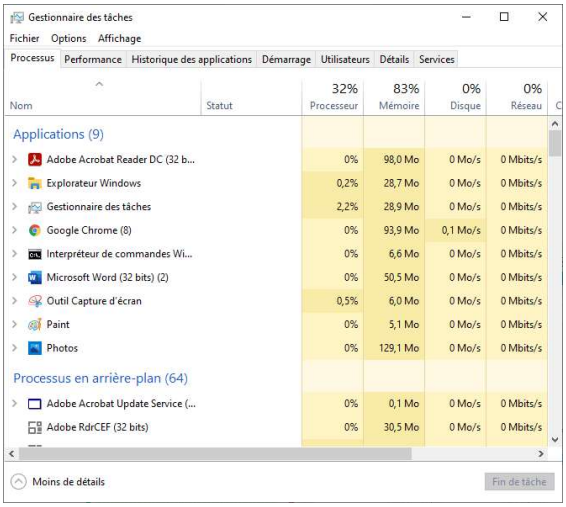
Dans le cas de Linux, le bootloader appelle la fonction **`start_kernel()`** du noyau.
Celle-ci s'occupe de détecter et d'initialiser le matériel, mais aussi de créer de toute pièce la structure de la tâche 0 (processus noyau de PID 0) en se définissant comme telle. Elle poursuit son exécution avec la fonction **`rest_init()`** qui va créer deux nouveaux processus en appelant :

- **`kernel_init()`** qui crée le processus **`/sbin/init (systemd)`** de PID 1, chargé de monter le système de fichiers et de démarrer le système d'exploitation sur le disque tout en passant en mode utilisateur.
- **`kthreadd()`** qui crée le processus **`[kthreadd]`** de PID 2, tournant en tâche de fond et qui est chargé de gérer les autres processus noyau.

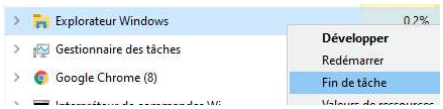
```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 09:55 ?        00:00:05 /sbin/init splash
root           2     0  0 09:55 ?        00:00:00 [kthreadd]
root           3     2  0 09:55 ?        00:00:00 [rcu_gp]
root           4     2  0 09:55 ?        00:00:00 [rcu_par_gp]
```

7. Sous Windows

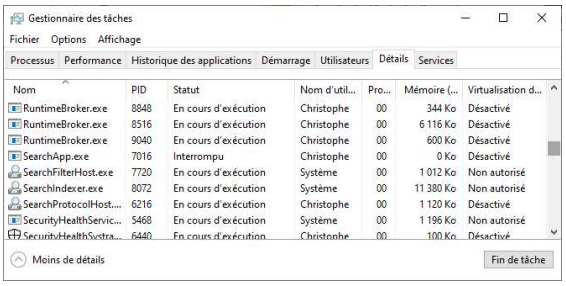
Sous Windows, le **gestionnaire des tâches** permet d'observer les processus en cours.



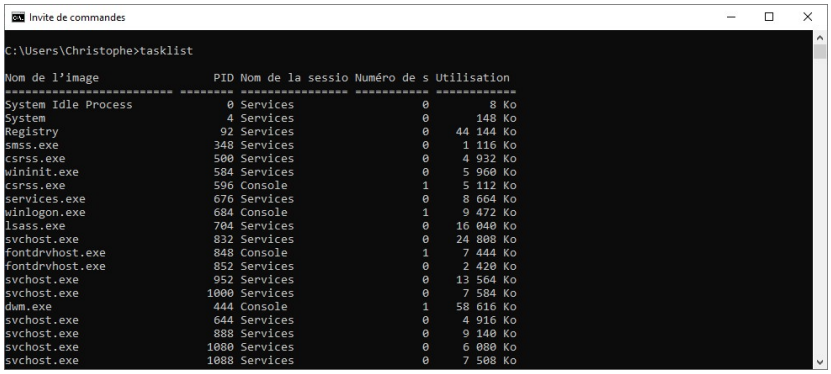
Un clic droit sur le processus permet de le fermer (**`Fin de tâche`**).



L'onglet **`Détails`** permet de connaître le PID de chaque processus.



Dans l'invite de commandes de Windows (**`cmd`**), la commande **`tasklist`** permet de visualiser les processus en cours.



La commande **`taskkill /PID 444`** permet de fermer le processus correspondant au programme **`dwm.exe`** (PID 444).

✍ Ouvrir le logiciel Paint, puis fermer son processus à l'aide de l'invite de commandes.

Source : https://www.lecluse.fr/nsi/NSI_T/archi/process/