1 Introduction

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**.
À 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**.

Nous avons tous été confrontés à la problématique de la gestion des processus dans un système d'exploitation, en tant qu'utilisateur :

- quand nous cliquons sur l'icône d'un programme, nous provoquons la naissance d'un ou plusieurs processus liés au programme que nous lançons.
- quand un programme ne répond plus, il nous arrive de lancer le gestionnaire de taches pour tuer le processus en défaut.

2 Notion de processus

Un **processus** est un **programme en cours d'exécution** sur un ordinateur.
Il est caractérisé par :

- un **ensemble d'instructions à exécuter** - souvent stockées dans un fichier sur lequel on clique pour lancer un programme (ex : firefox.exe)
- un **espace mémoire dédié** à ce processus pour lui permettre de travailler sur des données qui lui sont propres (ex : si vous lancez deux instances de firefox, chacune travaillera indépendamment de l'autre avec ses propres données.)
- des **ressources matérielles** : processeur, entrées-sorties (ex : accès à internet en utilisant la connexion Wifi).

Il ne faut donc pas confondre le fichier contenant un programme et le ou les processus qu'ils engendrent quand ils sont exécutés : un programme est juste un fichier contenant une suite d'instructions (ex : firefox.exe) alors que les processus sont des instances de ce programme ainsi que les ressources nécessaires à leur exécution (ex : plusieurs fenêtres de Firefox ouvertes en même temps).

3 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)

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

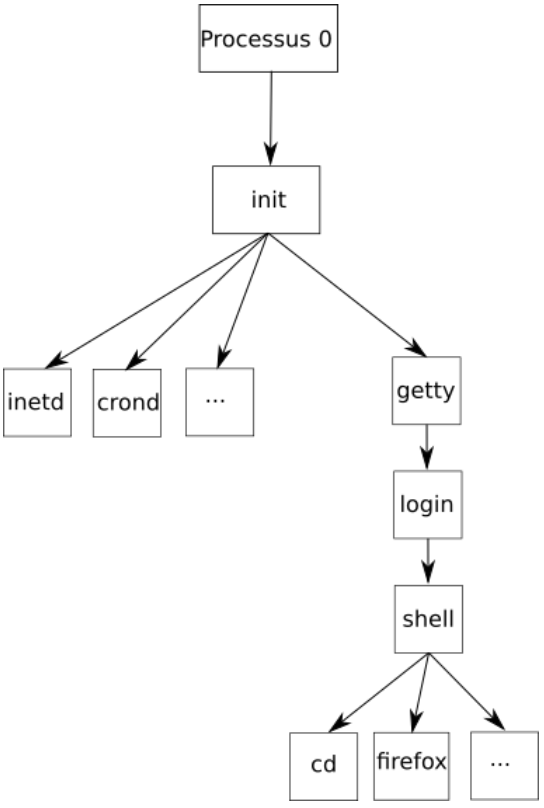
- le processus qui fait appel à fork() est appelé **processus père**.
- le processus qui est ainsi créé par clonage est le **processus fils**.
- après le clonage, un processus peut remplacer son programme par un autre programme grâce à l'*appel système* exec().

4 Arborescence de processus

Ce système de création un peu particulier (désigné souvent par fork/exec) conduit à l'émergence d'une arborescence de processus : un processus père engendre un ou plusieurs fils qui à leur tour engendrent d'autres fils, etc...

Si un processus est créé à partir d'un autre processus, comment est créé le tout premier processus ?
Sous un système d'exploitation comme Linux, au moment du démarrage de l'ordinateur un 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). À partir de init, les processus nécessaires au bon fonctionnement du système sont créés (par exemple les processus crond, inetd, getty,...). Puis d'autres processus sont créés à partir des fils de init...

On peut résumer tout cela avec le schéma suivant :



Remarque : Tous ces noms de processus ne sont pas à retenir, ils sont juste donnés pour l'exemple. Il est juste nécessaire d'avoir compris les notions de processus père et processus fils et la structure arborescente.

5 PID, PPID

Chaque processus possède un identifiant appelé PID (Process Identification), ce PID est un nombre.

Le premier processus créé au démarrage du système à pour PID 0, le second 1, le troisième 2... Le système d'exploitation utilise un compteur qui est incrémenté de 1 à chaque création de processus, le système utilise ce compteur pour attribuer les PID aux processus.

Chaque processus possède aussi un PPID (Parent Process Identification).

Ce PPID permet de connaître le processus parent d'un processus (par exemple le processus init vu ci-dessus à un PID de 1 et un PPID de 0). À noter que le processus 0 ne possède pas de PPID (c'est le seul dans cette situation).

6 Gérer les processus sur un système Linux

6.1 Commandes ps et top

Il est possible de les visualiser grâce à la commande `ps`.
Pour un affichage plus complet, page par page, utiliser `ps -ef | more`.

La commande `ps` ne permet pas de suivre en temps réel les processus (affichage figé). Pour avoir un suivi en temps réel, il faut utiliser la commande `top`.

Cette commande donne quelque chose du genre :

```
top - 17:38:22 up  8:59,  1 user,  load average: 0,58, 0,23, 0,23
Tâches: 335 total,   1 en cours, 264 en veille,   0 arrêté,   0 zombie
%Cpu(s): 19,7 ut,  2,8 sy,  0,0 ni, 77,2 id,  0,0 wa,  0,0 hi,  0,3 si,  0,0 st
KiB Mem :  8043552 total,   631412 libr,  4138584 util,  3273556 tamp/cache
KiB Éch:  2097148 total,  2095800 libr,     1348 util.  2820276 dispo Mem

  PID UTIL.      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TEMPS+ COM.
 3342 profjah+  20   0 2485884 447864  32832 S 183,4  5,6   2:54.80 nautilus
 1776 profjah+  20   0 4487564 480060 113360 S  13,2  6,0  21:36.01 gnome-shell
 1474 profjah+  20   0 1293652 222356 183320 S   3,0  2,8  16:12.93 Xorg
 1802 profjah+   9 -11 3171552  19012  14292 S   1,0  0,2  17:01.19 pulseaudio
18941 profjah+  20   0 3369160 480436 198740 S   0,7  6,0   2:30.74 Web Content
26577 profjah+  20   0   45760   4160   3284 R   0,7  0,1   0:00.35 top
    1 root      20   0  225960   8604   5624 S   0,3  0,1   0:22.70 systemd
    8 root      20   0       0      0      0 I   0,3  0,0   0:13.91 rcu_sched
 2307 profjah+  20   0  358404   9596   4940 S   0,3  0,1   2:39.29 ibus-daemon
 2422 profjah+  20   0  188140   4744   3880 S   0,3  0,1   0:00.54 dconf-service
 2698 profjah+  20   0 1443180  48564  31220 S   0,3  0,6   0:25.08 nautilus-deskto
 7192 profjah+  20   0 4234280 490040 143804 S   0,3  6,1  41:25.06 firefox
 7313 profjah+  20   0 2946888 175740  74012 S   0,3  2,2   7:51.80 WebExtensions
26397 root      20   0       0      0      0 I   0,3  0,0   0:00.03 kworker/4:1
```

L'affichage se rafraîchit en temps réel contrairement à ps qui fait un instantané. L'application est plus riche qu'il n'y paraît. Il faut passer un peu de temps à explorer toutes les options.

Celles-ci s'activent par des raccourcis clavier. En voici quelques unes :
- h : affiche l'aide.
- M : trie la liste par ordre décroissant d'occupation mémoire. Pratique pour repérer les processus trop gourmands.
- P : trie la liste par ordre décroissant d'occupation processeur.
- i : filtre les processus inactifs. Cela ne montre que ceux qui travaillent réellement.
- k : permet de tuer un processus - à condition d'en être le propriétaire.
  Essayez de tuer init ...
- q : permet de quitter top.

Remarque : Sous Windows, le gestionnaire de tâches (Ctrl+Alt+Suppr) donne des informations similaires.

6.2 Terminer un processus

Pour tuer un processus, on lui envoie un signal de terminaison. On en utilise principalement 2 :
- SIGTERM (15) : demande la terminaison d'un processus. Cela permet au processus de se terminer proprement en libérant les ressources allouées.
- SIGKILL (9) : demande la terminaison immédiate et inconditionnelle d'un processus. C'est une terminaison violente à n'appliquer que sur les processus récalcitrants qui ne répondent pas au signal SIGTERM.

La commande s'écrit ainsi (pour terminer le processus de PID 1493) :

```
kill -15 1493
kill -9 1493
```

7 Ordonnancement des processus par l'OS

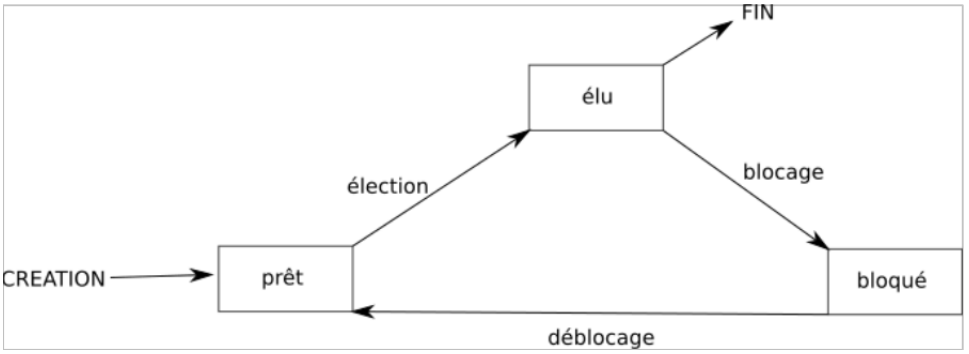
Dans un système multitâche plusieurs processus sont actifs simultanément, mais un processeur ne peut exécuter qu'une instruction à la fois. **Il va donc falloir partager le temps de processeur disponible entre tous les processus : c'est le travail de l'ordonnanceur** (ou scheduler en anglais). Ce dernier a pour tâche de sélectionner le processus suivant à exécuter parmi ceux qui sont prêts.
Tous les systèmes d'exploitation « modernes » sont donc capables de gérer l'exécution de plusieurs processus en même temps. Mais pour être précis, cela n'est pas en véritable « en même temps », mais plutôt un « chacun son tour ». Pour gérer ce « chacun son tour », les systèmes d'exploitation attributs des « états » au processus.

Un processus peut ainsi se trouver dans 3 états principaux :
- **prêt** (ready) : le processus attend son tour pour prendre la main.
- **en exécution, ou élu** (running) : le processus a accès au processeur pour exécuter ses instructions.
- **en attente, ou bloqué** (sleeping) : le processus attend qu'un événement se produise (saisie clavier, réception d'une donnée par le réseau ou le disque dur ...)

Il existe 2 autres états moins fondamentaux :

- *arrêté* (stopped) : le processus a fini son travail ou a reçu un signal de terminaison. Il libère les ressources qu'il occupe.
- *zombie* : une fois arrêté, le processus informe son parent afin que ce dernier l'élimine de la table des processus. Cet état est donc temporaire mais il peut durer si le parent meure avant de pouvoir effectuer cette tâche. Dans ce cas, le processus fils reste à l'état zombie...

Les 3 premiers états sont les plus importants puisqu'ils décrivent le cycle de vie normal d'un processus qui peut se résumé par le diagramme suivant :



1. Lorsqu'un processus est en train de s'exécuter (qu'il utilise le microprocesseur), il est dans l'état élu.
2. **Un processus qui se trouve dans l'état élu peut demander à accéder à une ressource** pas forcément disponible instantanément (par exemple lire une donnée sur le disque dur). **Le processus ne peut pas poursuivre son exécution tant qu'il n'a pas obtenu cette ressource.** En attendant de recevoir cette ressource, il passe de l'état élu à l'**état bloqué**.
3. Lorsque le processus finit par obtenir la ressource attendue, celui-ci peut potentiellement reprendre son exécution. Mais comme nous l'avons vu ci-dessus, les systèmes d'exploitation permettent de gérer plusieurs processus « en même temps », mais **un seul processus peut se trouver dans un état élu**. Quand un processus passe d'un état élu à un état bloqué, un autre processus peut alors « prendre sa place » et passer dans l'état élu. Le processus qui vient de recevoir la ressource attendue ne va donc pas forcément pouvoir reprendre son exécution tout de suite...
4. Un processus qui quitte l'état bloqué ne repasse pas forcément à l'état élu, il peut, en attendant que la place se libère passer dans l'état prêt (sous entendu "j'ai obtenu ce que j'attendais, je suis prêt à reprendre mon exécution dès que la place sera libérée").

Le passage de l'état prêt vers l'état élu constitue l'opération d'**élection**. Le passage de l'état élu vers l'état bloqué est l'opération de **blocage**.

Un processus est toujours créé dans l'état prêt. Pour se terminer, un processus doit obligatoirement se trouver dans l'état élu.

Il est vraiment important de bien comprendre que le « chef d'orchestre » qui attribue aux processus leur état élu, bloqué ou prêt est le système d'exploitation : il gère l'**ordonnancement** des processus.

Afin d'élire quel processus va repasser en mode exécution, l'ordonnanceur applique un algorithme prédéfini lors de la conception de l'OS. Le choix de cet algorithme va impacter directement la réactivité du système et les usages qui pourront en être fait. C'est un élément critique du système d'exploitation.

Sous Linux, on peut passer des consignes à l'ordonnanceur en fixant des priorités aux processus dont on est propriétaire : Cette priorité est un nombre entre -20 (plus prioritaire) et +20 (moins prioritaire).

Dernière chose à noter : un processus qui utilise une ressource R doit la libérer une fois qu'il a fini de l'utiliser afin de la rendre disponible pour les autres processus. Pour libérer une ressource, un processus doit obligatoirement être dans un état élu.

8 Interblocage (deadlock)

Les interblocages sont des situations de la vie quotidienne. Un exemple est celui du carrefour avec priorité à droite où chaque véhicule est bloqué car il doit laisser le passage au véhicule à sa droite.

En informatique également, **l'interblocage peut se produire lorsque des processus concurrents s'attendent mutuellement**. Les processus bloqués dans cet état le sont définitivement.

> L'interblocage peut se produire dans un environnement où des ressources sont partagées entre plusieurs processus et l'un d'entre eux détient indéfiniment une ressource nécessaire pour l'autre.

Mise en situation : Soit 2 processus P_1 et P_2, soit 2 ressources R_1 et R_2.

- Initialement, les 2 ressources sont libres (utilisées par aucun processus).
- Le processus P_1 commence son exécution (état élu), il demande la ressource R_1. Il obtient satisfaction puisque R_1 est libre, P_1 est donc dans l'état prêt.
- Pendant ce temps, le système a passé P_2 à l'état élu : P_2 commence son exécution et demande la ressource R_2. Il obtient immédiatement R_2 puisque cette ressource est libre.
- P_2 repasse immédiatement à l'état élu et poursuit son exécution.
- P_2 demande la ressource R_1, il se retrouve dans un état bloqué puisque la ressource R_1 a été attribuée à P_1 (P_1, dans l'état prêt, n'a pas eu l'occasion de libérer cette ressource puisqu'il n'a pas eu l'occasion de l'utiliser en repassant à l'état élu).
- P_2 étant bloqué (en attente de R_1), le système passe P_1 dans l'état élu mais avant de libérer R_1, P_1 demande à utiliser R_2. Problème ! R_2 n'a pas encore été libéré par P_2 ; cette ressource étant donc indisponible, P_1 se retrouve bloqué.

Résumons la situation à cet instant : P_1 possède la ressource R_1 et se trouve dans l'état bloqué (attente de R_2) ; P_2 pour sa part possède la ressource R_2 et se trouve dans l'état bloqué (attente de R_1).
Pour que P_1 puisse poursuivre son exécution, il faut que P_2 libère la ressource R_2, mais P_2 ne peut pas poursuivre son exécution (et donc libérer R_2) puisqu'il est bloqué dans l'attente de R_1. Pour que P_2 puisse poursuivre son exécution, il faut que P_1 libère la ressource R_1, mais P_1 ne peut pas poursuivre son exécution (et donc libérer R_1) puisqu'il est bloqué dans l'attente de R_2. Bref, la situation est totalement bloquée !

Remarque : Il existe des solutions permettant soit de mettre fin à un interblocage (cela passe par l'arrêt d'un des 2 processus fautifs) ou d'éviter les interblocages, mais ces solutions ne seront pas étudiées ici.

9 Complément

Révision de 1ère sur les commandes UNIX :
`http://ujuridec.free.fr/Terminus/` ou `http://luffah.xyz/bidules/Terminus/`
Une version peut-être plus lente que l'autre... à tester.

Série d'articles sur Interstices.info :
Le système d'exploitation, pièce centrale de la sécurité (2020) :
`https://interstices.info/le-systeme-dexploitation-piece-centrale-de-la-securite/`

À quoi sert un système d'exploitation ? (2015) :
`https://interstices.info/a-quoi-sert-un-systeme-dexploitation/`

Le ballet des processus dans un système d'exploitation (2015) :
`https://interstices.info/le-ballet-des-processus-dans-un-systeme-dexploitation/`

Alan Turing : du concept à la machine (2012) : (vérifier la pretinence)
`https://interstices.info/alan-turing-du-concept-a-la-machine/`

Sous le signe du calcul (2012) : (vérifier la pretinence)
`https://interstices.info/sous-le-signe-du-calcul/`