	**Architecture Matérielle**	**NSI T^{ale}**
	Les Processus	Cours/TD

*

1. Notion de processus

Un programme écrit à l'aide d'un langage de haut de niveau (on parle de "code source") est, comme nous l'avons vu l'année dernière, transformé en langage machine afin de pouvoir être exécuté par un ordinateur.

On appelle **processus** **un programme en cours d'exécution**. Attention, il ne faut pas confondre le code source du programme et un processus, qui lui correspond à l'exécution de ce programme par un ordinateur. Pour prendre une image assez classique, si une recette de cuisine correspond au code source du programme, le cuisinier en train de préparer cette recette dans sa cuisine correspond à un processus.

2. États d'un processus

Tous les **systèmes d'exploitation** "modernes" (Linux, Windows, macOS, Android, iOS...) sont 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.

Voici les différents **états** :

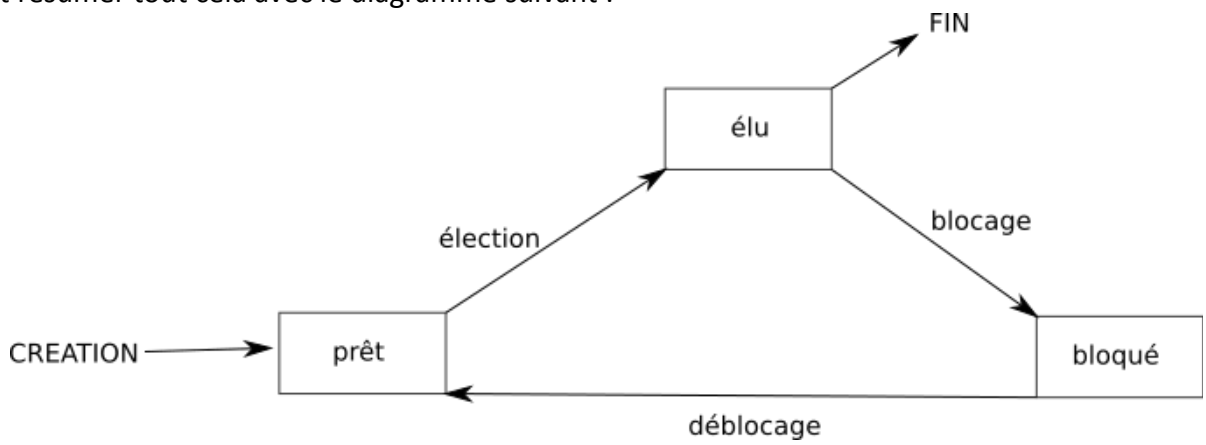
- Lorsqu'un processus est **en train de s'exécuter** (qu'il utilise le microprocesseur), on dit que le processus est dans l'**état "élu"**.

- 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"**

- 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" (le microprocesseur ne peut "s'occuper" que d'un seul processus à la fois). 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, car pendant qu'il était dans à état "bloqué" un autre processus a "pris sa place". 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"**.

On peut résumer tout cela avec le diagramme suivant :



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. On dit que **le système gère l'ordonnancement des processus** (tel processus sera prioritaire sur tel autre...)

Chose aussi à ne pas perdre de vu : 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"**.

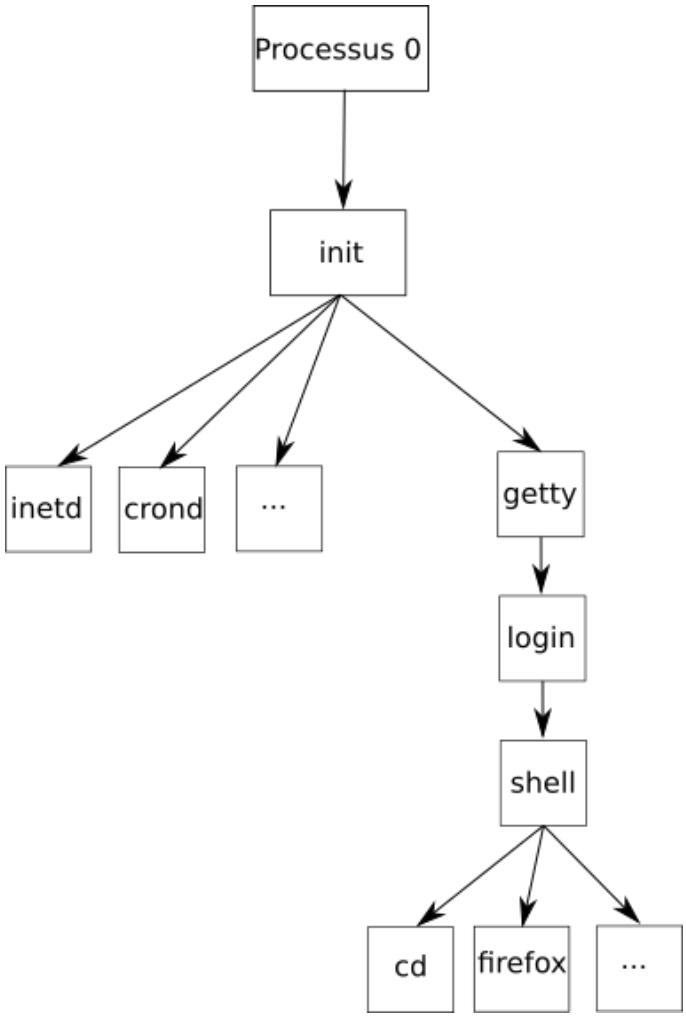
3. Création d'un processus

Un processus peut créer un ou plusieurs processus à l'aide d'une commande système ("fork" sous les systèmes de type Unix).
Imaginons un processus A qui crée un processus B. On dira que A est le père de B et que B est le fils de A. B peut, à son tour créer un processus C (B sera le père de C et C le fils de B). On peut modéliser ces relations père/fils par une structure arborescente (voir le cours si nécessaire).

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 :



N.B. 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.

4. PID et PPID

Chaque processus possède un identifiant appelé **PID** (***Process Identifier***), ce **PID est un nombre**.
Le premier processus créé au démarrage du système a 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 Identifier***).
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).

À faire vous-même 1

- En vous basant sur le schéma ci-dessus, donnez le PID (en partant du principe qu'il est créé juste après init) et le PPID du processus "getty".

5. Observer les processus

Sous Linux il existe des commandes permettant de visualiser les processus.

À faire vous-même 2

- Après avoir ouvert un terminal, tapez la commande suivante : `ps -ef`

Vous devriez avoir plusieurs informations sur les processus en cours sur votre ordinateur, notamment les PID et les PPID de ces processus.

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

À faire vous-même 3

- Après avoir ouvert un terminal, tapez la commande suivante : `top`

- Pour en savoir plus sur la commande top, consultez la page http://debian-facile.org/doc:systeme:top

À faire vous-même 4

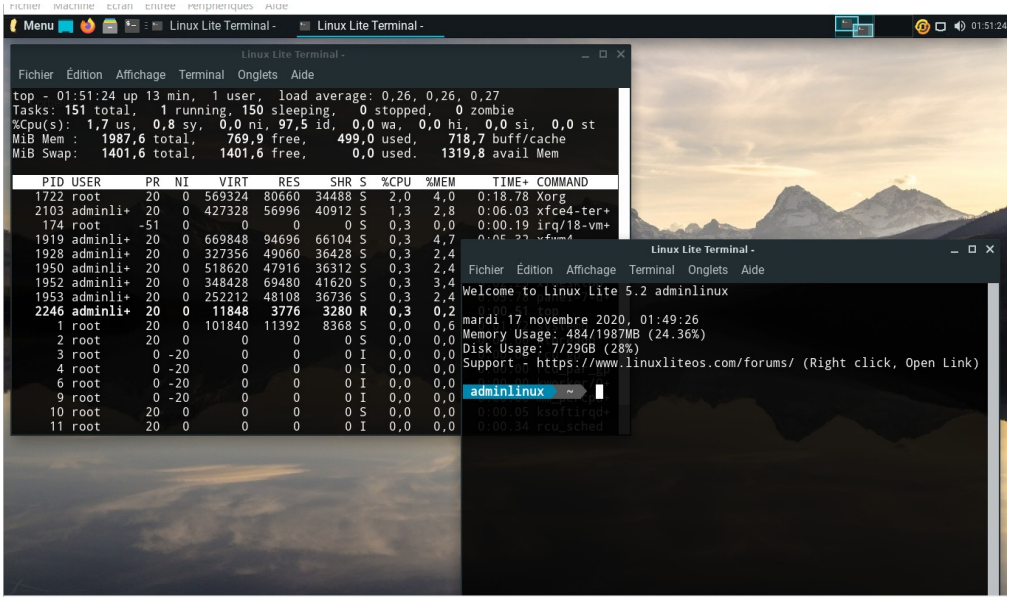
- En utilisant la commande `top` dans un terminal, observez ce qui se passe au niveau des processus quand vous ouvrez ou supprimez un onglet dans votre navigateur web.

Il est possible de supprimer un processus en utilisant la commande `kill`. L'utilisation de cette commande est très simple, il suffit de taper `kill` suivi du PID du processus à supprimer (exemple : `kill 4242` permet de supprimer le processus de PID 4242)

À faire vous-même 5

Ouvrez 2 terminaux, placez-les l'un à côté de l'autre. Dans l'un des 2 terminaux, exécutez la commande `top`

Vous devriez obtenir quelque chose qui ressemble à ceci :



- Fermez le deuxième terminal et observez le résultat dans celui exécutant `top`.

- Ouvrez votre navigateur Web et observez le résultat dans le terminal exécutant `top`. Selon le navigateur que vous utilisez, il se peut que le fonctionnement du navigateur soit associé à plusieurs processus.

- Notez le PID des processus liés au fonctionnement du navigateur.

- Utilisez la commande `kill` afin de supprimer le (ou les) processus lié(s) au fonctionnement du navigateur.
- Que se passe-t-il ?

6. Interblocage

Pour terminer, nous allons maintenant étudier le phénomène d'**interblocage** (***deadlock*** en anglais).

Soient **2 processus P1 et P2**, soient **2 ressources R1 et R2**.
Initialement, les 2 ressources sont "libres" (utilisées par aucun processus).
Le processus P1 commence son exécution (état élu), il demande la ressource R1. Il obtient satisfaction puisque R1 est libre, P1 est donc dans l'état "prêt".
Pendant ce temps, le système a passé P2 à l'état élu : P2 commence son exécution et demande la ressource R2. Il obtient immédiatement R2 puisque cette ressource était libre. P2 repasse immédiatement à l'état élu et poursuit son exécution (P1 lui est toujours dans l'état prêt). P2 demande la ressource R1, il se retrouve dans un état bloqué puisque la ressource R1 a été attribuée à P1 : P1 est dans l'état prêt, il n'a pas eu l'occasion de libérer la ressource R1 puisqu'il n'a pas eu l'occasion d'utiliser R1 (pour utiliser R1, P1 doit être dans l'état élu).
P2 étant bloqué (en attente de R1), le système passe P1 dans l'état élu et avant de libérer R1, il demande à utiliser R2. Problème : R2 n'a pas encore été libéré par P2, R2 n'est donc pas disponible, P1 se retrouve bloqué.

Résumons la situation à cet instant : **P1 possède la ressource R1 et se trouve dans l'état bloqué** (attente de R2), **P2 possède la ressource R2 et se trouve dans l'état bloqué** (attente de R1)

Pour que P1 puisse poursuivre son exécution, il faut que P2 libère la ressource R2, mais P2 ne peut pas poursuivre son exécution (et donc libérer R2) puisqu'il est bloqué dans l'attente de R1. Pour que P2 puisse poursuivre son exécution, il faut que P1 libère la ressource R1, mais P1 ne peut pas poursuivre son exécution (et donc libérer R1) puisqu'il est bloqué dans l'attente de R2. Bref, la situation est totalement bloquée !

Cette situation est qualifiée d'**interblocage** (***deadlock*** en anglais).

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

À faire vous-même 6
Mettez au point une petite saynète de théâtre permettant d'expliquer l'interblocage de 2 processus. Par exemple on pourra avoir 2 acteurs qui joueront le rôle des processus, des étiquettes qui représenteront les différents états (élu, bloqué ou prêt) des processus et des étiquettes qui représenteront les ressources R1 et R2.

Source : https://pixees.fr/informatiquelycee/n_site/nsi_term_archi_proc.html