Séq. 17 – Gestion des processus et des ressources par un OS

Objectifs

1. Décrire la création d'un processus
2. Décrire l' ordonnancement de plusieurs processus par le système
3. Mettre en évidence le risque de l'interblocage (deadlock)

Cette séquence s'appuie sur :

- https://pixees.fr/informatiquelycee/n_site/nsi_term_archi_proc.html
- https://lecluseo.frama.io/leclusemaths/nsi/NSI_T/archi/process/

1 Introduction

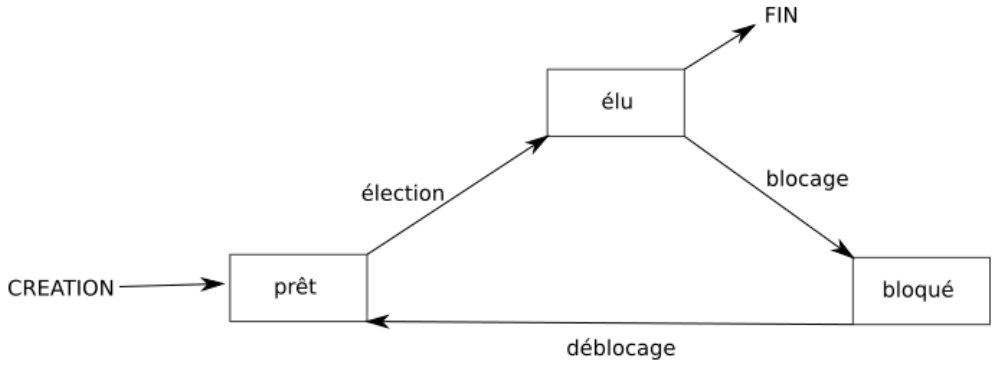
Un processus est un programme en cours d'exécution sur un ordinateur. Il est caractérisé par :

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

Il ne faut donc 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.
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 (Linux, Windows, macOS, Android, iOS...) sont multi-tâches et sont donc capables de gérer l'exécution de plusieurs processus en même temps.
Mais 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 attribuent des "états" au processus.
On peut résumer tout cela avec le diagramme suivant :



2.1 État « prêt »

Un processus est toujours créé dans l'état "prêt".

2.2 État « élu »

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

2.3 État « bloqué »

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 un état "bloqué" un autre processus a "pris sa place".

2.4 Changements d' état

Un processus qui quitte l'état bloqué ne repasse pas forcément à l'état "élu", il peut, en attendant que "la place se libère" repasser 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".
Pour se terminer, un processus doit obligatoirement se trouver dans l'état "élu".

2.5 Rôle de l' OS

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...)

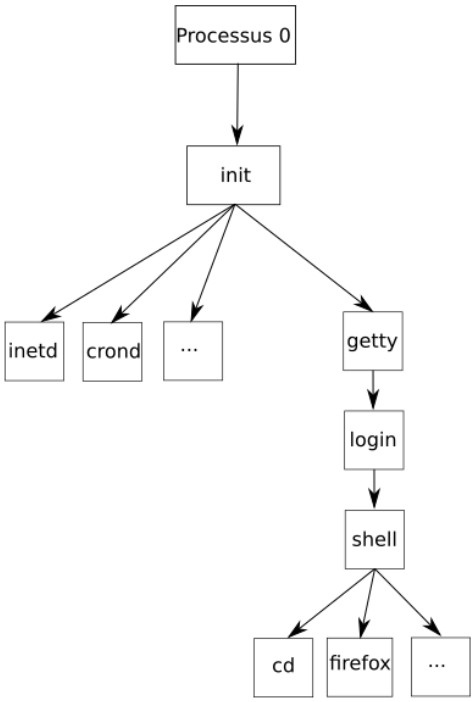
Chose aussi à ne pas perdre de vu : un processus qui utilise une ressource R doit la "libérer" une fois qu'il a fini de l'utiliser afin de la rendre disponible pour les autres processus. Pour libérer une ressource, un processus doit obligatoirement être dans un état "élu".

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 créé 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.

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 ci-contre :

N. B. Tous ces noms de processus ne sont pas à retenir.



A faire vous même 1.

- Connectez-vous à un système linux comme un Raspberry pi
- Ouvrez un terminal et tapez la commande :
  `$ pstree`
- Reproduisez les éléments intéressants de l' arborescence :

4 Identification des processus - PID et 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).

A faire vous même 2.

- Ouvrez un terminal et tapez la commande :
  `$ ps -eF`
- Quel est le processus associé au PID le plus petit ?
- Quel est son PPID ?
- Citer quelques descendants directs de ce processus.

A faire vous même 3.

- Ouvrez un terminal et tapez la commande :
  `$ top`
- Allez sur les pages web :
  https://debian-facile.org/doc:systeme:top
  https://www.tremplin-numerique.org/comment-utiliser-la-commande-linux-top-et-comprendre-sa-sortie
- Testez les différentes possibilités de `top`
- Exécutez un navigateur web. Quel est son PID ? Son PPID ?
- Ouvrez un 2e terminal. Que se passe-t-il ?
- Observez ce qui se passe au niveau des processus quand vous ouvrez ou supprimez un onglet dans votre navigateur web.
- Lancez un script python avec une récursivité explosive (Suite de Fibonacci par exemple). Observez le processus associé. Relevez son PID.
- Pour arrêter ou « tuer » le processus, on utilise la commande `kill` :
  `$ kill -15 PID`

- Lire P. 234-235

5 Interblocage

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

Cette situation d'interblocage a été théorisée par l' informaticien Edward Coffman (1934-) qui a énoncé quatre conditions (appelées conditions de coffman) menant à l'interblocage :

- Exclusion mutuelle : au moins une des ressources du système doit être en accès exclusif.
- Rétention des ressources : un processus détient au moins une ressource et requiert une autre ressource détenue par un autre processus
- Non préemption : Seul le détenteur d'une ressource peut la libérer.
- Attente circulaire : Chaque processus attend une ressource détenue par un autre processus. P_1 attend une ressource détenue par P_2 qui à son tour attend une ressource détenue par P_3 etc... qui attend une ressource détenue par P_1 ce qui clos la boucle.

Il existe heureusement des stratégies pour éviter ces situations. Nous ne rentrerons pas ici dans ces considérations qui dépassent le cadre du programme.

Exemple détaillé :
Soit 2 processus P1 et P2, soit 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.

- Lire P. 236-237

- Vidéo1 - Le système d'exploitation en trois idées clé : https://www.youtube.com/watch?v=SpCP2oaCx8A
- Vidéo2 - Processus dans un système d'exploitation : https://www.youtube.com/watch?v=bFqud0gcCHM ☐

A faire vous même 4. ☐

- P. 242 ex 2
- P. 242 ex 4
- P. 242 ex 5
- P. 242 ex 8

A faire vous même 5. Pour les rapides ☐

- P. 242 ex 6
- P. 242 ex 7
- P. 242 ex 9

A faire vous même 6. Pour les rapides et les plus motivés ☐

- P. 242 ex 11