Gestion des processus

1. Gestion des processus

Au tout début, les processeurs étaient monotâche : ils ne pouvaient gérer qu'un seul processus à la fois. Dans les années 1960, les processeurs multitâches se sont généralisés. L'utilisateur à l'impression que plusieurs processus s'exécutent simultanément. Ce n'est pas le cas en général, les processus sont exécutés par petits segments en rotation très rapide dans le processeur. Certains logiciels, sur des processeurs multicœurs, sont réellement exécutés en parallèle.

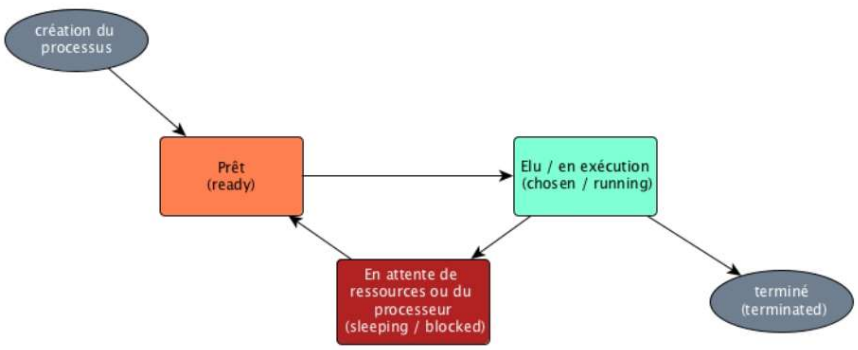
Une des tâches du système d'exploitation est d'allouer à chacun des processus les ressources dont il a besoin en termes de mémoire, entrées-sorties ou temps processeur, et de s'assurer que les processus ne se gênent pas les uns les autres.
En effet, il doit permettre à toutes les applications et tous les utilisateurs de travailler en même temps, c'est-à-dire donner l'impression à chacun qu'il est seul à utiliser l'ordinateur et ses ressources physiques. Cette gestion complexe des processus est réalisée par une partie spécifique du noyau : l'**ordonnanceur**.
Comme une ressource (le processeur ou un périphérique) ne peut pas être partagée, c'est son temps d'utilisation qui va l'être : le temps d'utilisation d'une ressource est partagé en intervalles très courts, pendant lesquels l'ordonnanceur l'alloue à un seul utilisateur.

2. État d'un processus

Lorsqu'un processus sollicite une ressource (par exemple la lecture d'un fichier) et que celle-ci n'est pas immédiatement accessible (le disque dur est déjà en cours d'utilisation), le processus va être mis en sommeil un certain temps pour être ensuite réveillé.
Ainsi, durant son existence, le système d'exploitation attribut des états au processus :

- Le processus est dans **l'état élu** lorsqu'il est en train de s'exécuter (il utilise le microprocesseur) ; un seul processus peut se trouver dans un état élu : le microprocesseur ne peut s'occuper que d'un seul processus à la fois.
- Le processus est dans **l'état bloqué** lorsque de l'état élu, il demande à accéder à une ressource qui n'est pas forcément disponible instantanément (par exemple lire une donnée sur le disque dur). Comme **le processus ne peut pas poursuivre son exécution tant qu'il n'a pas obtenu cette ressource**, il passe de l'état élu à l'état bloqué.
- Le processus est dans **l'état prêt** lorsqu'il finit par obtenir la ressource attendue mais qu'il **ne peut pas forcément reprendre son exécution immédiatement** car un autre processus est passé dans l'état élu à sa place pendant qu'il était dans à état bloqué. Cet état signifie : "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**.



Le système d'exploitation attribue aux processus leur état élu, bloqué ou prêt. On dit que le système gère **l'ordonnancement des processus** (tel processus sera prioritaire sur tel autre...).

Remarque : 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. Multitâche et ordonnancement

Un système d'exploitation est dit multitâche lorsqu'il est capable de simuler l'exécution simultanée de plusieurs processus indépendamment du nombre de cœurs disponibles dans le microprocesseur. Pour mettre en œuvre le multitâche, deux approches sont possibles :

- Le **multitâche coopératif** qui consiste à laisser les processus décider du moment où ils doivent rendre la main aux autres. Celui-ci pose cependant deux principales difficultés : il faut que le multitâche soit pris en compte lors de l'écriture des programmes et en cas d'erreur le système tout entier peut se retrouver bloqué. C'est ce type de multitâche que l'on retrouve dans Windows 3.11 (1993) ou Mac OS 9 (1999).
- Le **multitâche préemptif** qui consiste à octroyer un certain temps d'exécution au processus avant de reprendre la main de force en sauvegardant l'état du processus, au moyen d'une interruption programmable. Ce type de multitâche se retrouve dans les systèmes d'exploitation Unix (1969) et Windows NT 3.1 (1993) mais aussi dans les systèmes plus grand-public comme AmigaOS (1985), Windows 95 (1995) et Mac OS X (2001).

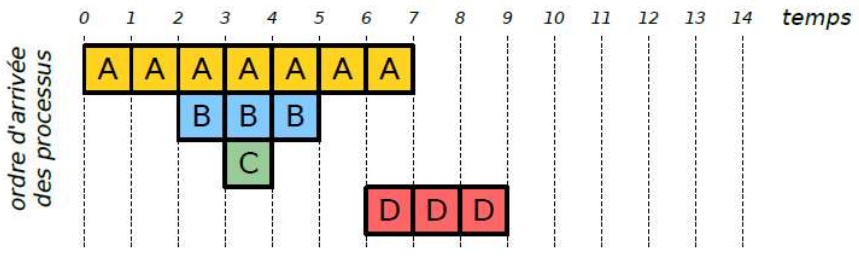
En accordant quelques millisecondes de temps d'exécution à chaque processus, un système d'exploitation multitâche préemptif comme Linux, doit effectuer de nombreuses commutations de contexte (sauvegardes et restauration des registres du processeur, changement de l'espace d'adressage…) avec un impact parfois non négligeable sur les performances du système. Il convient alors de gérer au mieux l'ordre et le temps d'exécution des processus en prenant en compte leurs priorités et l'accès aux ressources du système, tout en maintenant une certaine équité entre les processus. On appelle ce pilotage l'**ordonnancement**.

L'ordonnanceur (scheduler) permet :

- de minimiser le temps de traitement du processus d'un utilisateur
- de garantir l'équité entre les différents utilisateurs
- d'optimiser l'utilisation de la ressource
- d'éviter les blocages

4. Algorithmes d'ordonnancement

Pour illustrer les différents algorithmes d'ordonnancement de tâches on peut considérer la situation simplifiée de quatre processus A, B, C et D de même priorité dont le moment d'arrivée est donné par l'échelle de temps et de durée indiquée par le découpage en cycles ou quantum de temps pleins (nombre entier de carrés).



4.1. La file (FIFO)

Le premier processus arrivé est le premier à être exécuté et ce n'est qu'à sa fin que le processus suivant peut commencer.
L'algorithme conduit alors à l'ordonnancement suivant :

La tâche A s'exécute entièrement avant que ne vienne le tour de la tâche B, puis celui de la tâche C et enfin celui de la tâche D. C'est l'ordre d'arrivée des tâches qui définit l'ordre d'exécution.

4.2. Le plus rapide à finir en premier (Shortest Remaining Time First)

On choisit d'exécuter en priorité le processus dont le temps d'exécution restant est le plus petit. Comme il est impossible de prévoir à l'avance ce temps d'exécution, on se base sur une estimation du temps restant par le calcul d'une moyenne des temps d'exécution réalisés précédemment.
L'algorithme conduit alors à l'ordonnancement suivant :

Lorsque arrive la tâche B, d'une durée de 3 cycles, la tâche A en a encore 5 à accomplir, elle cède donc la place à la tâche B. Puis la tâche C survient avec une durée de 1 cycle inférieure à celle de B à qui il reste encore 2 cycles à ce moment-là. C'est donc la tâche C qui est exécutée. Puis la tâche B reprend et se termine à l'instant où la tâche D arrive avec ses 3 cycles et passe donc devant la tâche A qui est reléguée tout à la fin en raison de ses 5 cycles restants à accomplir.

4.3. Le tourniquet (Round-Robin)

On attribue un quantum de temps identique à chaque tâche extraite de la file d'attente, pour l'y replacer dès que le quantum est échu.
L'algorithme conduit alors à l'ordonnancement suivant :

Explications :

0. A étant la seule tâche élue au départ, la file d'attente est vide.
1. A reste ainsi élue au deuxième quantum.
2. La tâche B arrive est insérée dans la file.
   On enfile A derrière B, et on extrait B pour lui attribuer un quantum.
3. La tâche C arrive et est insérée dans la file derrière A.
   On enfile B derrière C, et on extrait A pour lui attribuer un quantum.
4. On enfile A derrière B, et on extrait C pour lui attribuer un quantum.
5. C'est terminé. La file contient les tâches B et A dans cet ordre.
   On défile B pour lui attribuer un quantum.
6. La tâche D arrive, elle est enfilée derrière la tâche A dans la file.
   On enfile B derrière D, et on extrait A pour lui attribuer un quantum.
7. On enfile A derrière B, et on extrait D pour lui attribuer un quantum.
8. On enfile D derrière A, et on extrait B pour lui attribuer un quantum.

9. B est terminée. La file contient les tâches A et D dans cet ordre.
   On défile A pour lui attribuer un quantum.
10. On enfile A et on extrait D pour lui attribuer un quantum.
11. On enfile D et on extrait A pour lui attribuer un quantum.
12. On enfile A et on extrait D pour lui attribuer un quantum.
13. D est terminée. La file ne contient que la tâche A.
    On extrait A pour lui attribuer un quantum.
14. A est terminée la file est vide.

4.4. Completely Fair Scheduler (CFS)

C'est l'ordonnanceur actuellement utilisé sous Linux depuis la version 2.6.23 sortie en 2007. Les tâches sont placées dans un arbre rouge-noir (un arbre binaire de recherche) sur la base d'un temps d'exécution virtuel. Cette structure de données sert à ordonner les tâches à exécuter, en permettant de les extraire en temps constant et de le réinsérer ensuite avec une complexité en O(log(n)).

Ce ne sont là que quelques exemples d'ordonnanceurs, il en existe beaucoup d'autres adaptées aux différents types de systèmes d'exploitation (multitâche coopératif, multitâche préemptif, systèmes temps-réel).

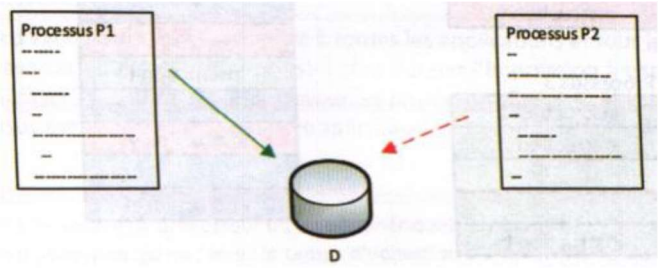
5. Interblocage (deadlock)

Des processus peuvent avoir besoin de la même ressource. Dans de nombreuses situations, deux processus (ou davantage) peuvent souhaiter accéder à la même donnée sur le disque dur :

- Les deux processus ont uniquement besoin de lire la donnée : celle-ci est alors partagée, sans problème complexe.
- Les deux processus ont besoin de la donnée de manière exclusive, pour la modifier par exemple.
- Les deux processus ont besoin de communiquer entre eux : l'un doit attendre un résultat de l'autre

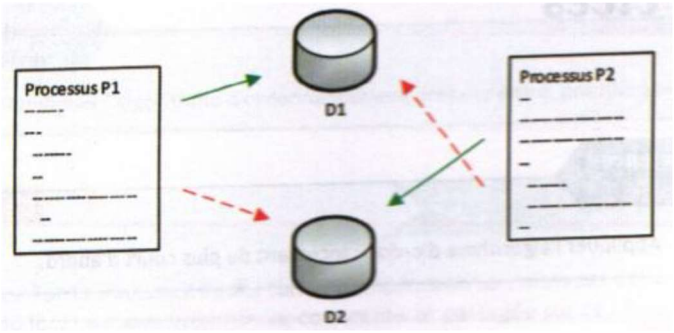
5.1. Premier exemple

Sur le schéma ci-dessous, les processus P1 et P2 ont tous les deux besoins de la même donnée D pour la modifier, c'est-à-dire de manière exclusive. Le premier à y accéder est P1, D lui est allouée par le système d'exploitation. Lorsque P2 souhaite accéder à D, la ressource n'est pas disponible : P2 est alors bloqué jusqu'à la fin de l'utilisation de D par Pl.



5.2. Deuxième exemple

Deux processus P1 et P2 ont tous les deux besoins de deux données, nommées sur le schéma suivant D1 et D2. Voici une situation qui peut se produire : **chaque processus bloque une donnée et est en attente de l'autre**, rien ne pourra évoluer sans une intervention extérieure : cette situation porte le nom **d'interblocage**.



5.3. Conditions d'interblocage

Edward G. Coffman Jr. a établi les conditions d'un interblocage :

- Au moins une ressource doit être conservée dans un mode non partageable.
- Un processus doit maintenir une ressource et en demander une autre.
- Une ressource ne peut être libérée que par le processus qui la détient.
- Chaque processus doit attendre la libération d'une ressource détenue par un autre qui fait de même.

Pour régler le problème de l'interblocage, quelques approches sont possibles :

- s'en remettre à l'utilisateur pour régler lui-même le problème.
- anticiper et éviter le problème (voir l'algorithme du banquier de E. Dijkstra)
- prévenir et empêcher l'apparition du problème en agissant sur les priorités et l'ordre d'exécution des processus.

6. Application

On considère 3 processus P1, P2 et P3 et 3 ressources R1, R2 et R3 tels que :

- P1 : demande R1, demande R2, libère R1, libère R2.
- P2 : demande R2, demande R3, libère R2, libère R3.
- P3 : demande R3, demande R1, libère R3, libère R1.

✍ Si les processus sont exécutés l'un après l'autre, d'abord P1 puis P2 et enfin P3, y-a-t-il interblocage ?

✍ Décrire une exécution des 3 processus qui conduit à une situation d'interblocage.

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