La gestion des processus

Processus

Un processus est un programme en cours d'exécution.

- Un programme est une suite d'instructions ; c'est du texte, un code statique.
- Le processus est un concept dynamique, il représente le déroulement d'une tâche faisant partie d'une application ou un programme système quelconque.

La notion de processus est essentielle pour décrire le fonctionnement des systèmes multiprogrammés aussi appelés multitâches ou plus simplement multi processus.

Parmi les avantages de la multiprogrammation, citons :

- Le fractionnement des applications qui peut en simplifier le développement,
- l'avantage pour l'utilisateur de savoir faire tourner plusieurs applications simultanément
- et surtout l'optimalisation de l'utilisation du (ou des) processeurs(s)

Un processeur n'est capable de traiter qu'un seul processus à la fois. Un sous-ensemble du système d'exploitation, appelé **ordonnanceur**, organise les tâches et les fait commuter tout à tour pour donner l'impression qu'elles s'exécutent toutes simultanément.

Le système d'exploitation conserve des informations sur chaque processus pour pouvoir les interrompre et les relancer selon ce que décide l'ordonnanceur. Ces informations regroupent entre autres :

- un numéro d'identification du processus (PID)
- l'état du processus, son compteur ordinal et les autres registres
- l'emplacement mémoire du code, des données et de la pile
- des pointeurs vers les ressources utilisées, fichiers, E/S, …
- et une quantité innombrable d'informations : pointeur vers le processus parent, priorités, compteur de threads, durée d'exécutions, informations d'attentes etc.

Toutes ces informations peuvent être regardées comme les composants d'un processeur virtuel agent d'exécution du processus.

Les niveaux d'ordonnancement des processus

L'ordonnancement est à envisager à trois niveaux : à court, à moyen et à long terme.

1. L'ordonnancement à long terme "*job scheduling*" (ordonnancement des travaux) décide des processus que le système peut mener en parallèle. Ils doivent être assez nombreux pour que le processeur soit inactif le plus rarement possible (quand tous les processus attendent des E/S) sans pour autant être trop abondant et saturer la mémoire principale du système.
   - Dans un traitement par lots, les processus attendent dans une file d'attente sur le disque jusqu'à ce que le *scheduler* ou ***ordonnanceur d'admission*** décide de les prendre en charge.
   - Dans un système interactif à temps partagé, le système accepte en principe toutes les requêtes de processus que les utilisateurs provoquent en lançant leurs applications.

   Une fois le processus admis dans le système, il n'en sort que lorsqu'il est terminé ou s'il est détruit par le système d'exploitation suite à une erreur grave ou à la demande de l'utilisateur (commande *kill* sous Unix ou via le gestionnaire des tâches sous Windows)

2. L'ordonnancement à moyen terme est assuré par ***l'ordonnanceur de mémoire*** aussi appelé permutateur ou *swapper*. Son rôle est de permuter les processus placés en mémoire et ceux qui, faute de place, ont été temporairement entreposés sur le disque.

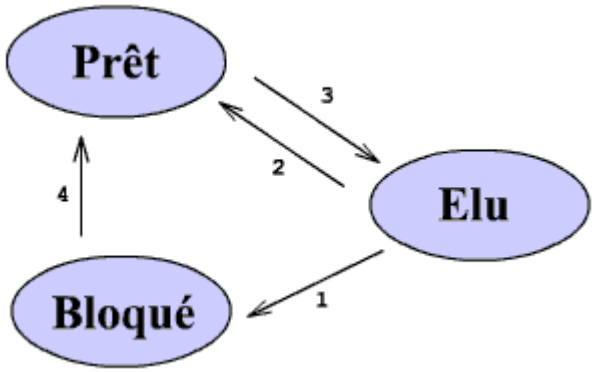
   Ces permutations ne peuvent toutefois pas être trop fréquentes pour ne pas gaspiller la bande passante des disques.

3. L'ordonnanceur à court terme aussi appelé *dispacher*, répartiteur ou ***ordonnanceur du processeur*** choisit à quel processus sera alloué le processeur et pour quel laps de temps.

   Ces commutations des processus sont très fréquentes.

Les états d'un processus

Les processus, puisqu'ils sont concurrents et doivent se partager le processeur, ne peuvent être continuellement actifs. Ils ont donc, si on ne considère pour commencer que l'ordonnancement à court terme, trois niveaux fondamentaux et quatre transitions possibles.



« **Elu** » signifie en cours d'exécution. L'exécution n'est interrompue que par les conditions suivantes :

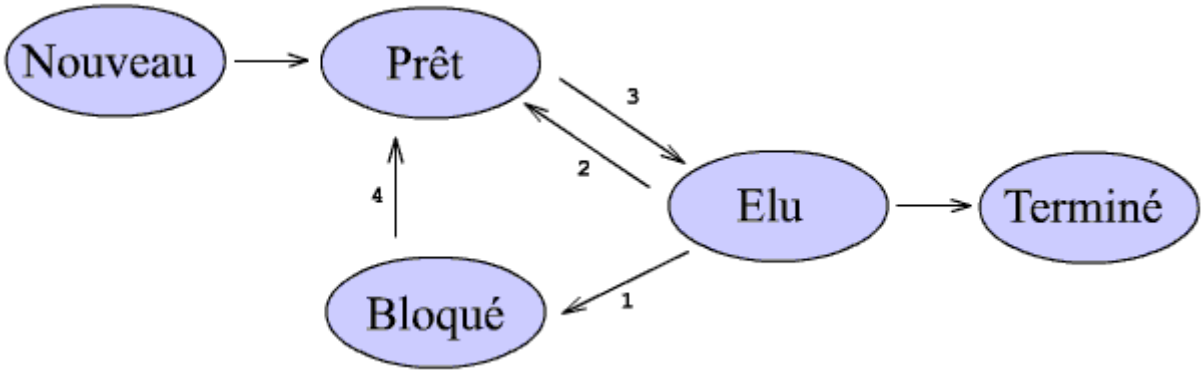
Transition 1 : Le processus se bloque, faute de données pour l'alimenter ou en attendant une opération d'entrée/sortie.

Transition 2 : Le processus est interrompu soit parce que la tranche de temps qui lui est impartie est achevée soit parce qu'un processus de plus haute priorité réquisitionne le processeur.

L'état « **Prêt** » est un état provisoire pour permettre aux autres processus de s'exécuter quasi simultanément.

L'état « **Bloqué** » est un état d'attente d'un événement extérieur, tel que l'acquisition de données nécessaires à la poursuite de l'exécution du processus.

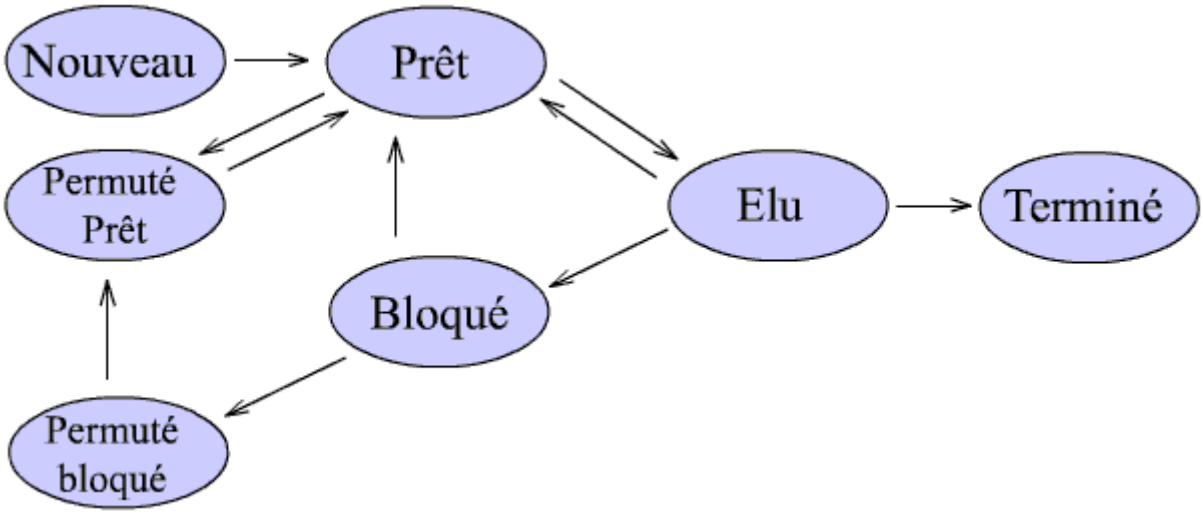
Ajoutons deux états qui correspondent à l'ordonnancement à long terme : Les états « **Nouveau** » et « **Terminé** ».



Nouveau : le processus vient d'être créé mais n'existe encore qu'à l'état de requête de processus en attendant d'être admis par le *scheduler* en tant que processus activable.

Terminé : le processus est désormais inactif car il a achevé sa tâche. Il sera détruit prochainement par le système d'exploitation pour libérer de la place en mémoire. Il est parfois conservé pendant un temps à l'état "Terminé" en attendant qu'une entrée/sortie s'achève ou que les données de ce processus soient reprises par un autre. On parle alors de processus "zombie".

Pour être complet il faut aussi envisager les états permutés qui résultent de l'ordonnancement à moyen terme. Le *swapper* range les processus prêts ou bloqués sur le disque ou en mémoire.



Permuté-Prêt : le processus est pour l'instant transcrit en mémoire auxiliaire (sur disque). Il serait prêt à être activé par l'ordonnanceur à court terme s'il était en mémoire principale. La permutation de mémoire dépend de l'ordonnanceur à moyen terme.

Permuté-Bloqué : c'est l'état d'un processus qui étant bloqué en attendant un événement externe à été transféré sur disque pour faire de la place en mémoire principale.

L'ordonnancement des processus

Les processus concurrents se partagent le processeur, la mémoire, les fichiers et les entrées/sorties.

- Dans les systèmes anciens et les systèmes d'exploitation pas vraiment multitâches tels que Windows 9x, l'ordonnancement était de type ***non préemptif***. L'ordonnanceur n'intervenait que lorsque le processus en cours se terminait, se bloquait ou se montrait assez coopératif pour volontairement rendre la main à l'ordonnanceur. Ce système assez sommaire convient aux traitements par lots quand le temps de réponse n'avait que peu d'importance.
- Actuellement, sur les systèmes interactifs multitâches et parfois même multi utilisateurs, l'ordonnancement doit être ***préemptif***. L'ordonnanceur ne peut laisser un processus monopoliser les ressources du système. Il réquisitionne régulièrement le processeur pour en répartir la disponibilité entre les processus qui simultanément sont prêts à être exécutés.

La politique suivie pour déterminer la manière d'ordonnancer les processus est fonction de nombreux critères parfois contradictoires. Le fait de favoriser certaines catégories de tâches peut en léser d'autres.

Critères d'ordonnancement

- L'utilisation intensive du processeur

Le système perd son efficacité si le processeur est occupé par des processus qui attendent la disponibilité d'une ressource.

- L'équité

Tous les processus doivent avoir la possibilité d'utiliser le processeur. Nonobstant les priorités parfois indispensables, des processus comparables doivent être traités avec les mêmes égards.

- Utilisation équilibrée de l'ensemble des ressources

En ne tenant compte que de l'utilisation optimale du CPU, on risquer de sous-utiliser momentanément d'autres ressources que les processus devront se disputer ensuite. Une bonne stratégie consiste donc à évaluer et à bien répartir l'emploi de l'ensemble des ressources.

Les objectifs cités ci-dessus sont difficilement mesurables. Il faut, pour bien faire, trouver des critères qui permettent de chiffrer l'efficacité d'une politique d'ordonnancement.
Les trois suivants s'appliquent plus particulièrement aux systèmes de traitement par lots.

- Le nombre de processus par unité de temps

Ce critère dépend cependant de la longueur des processus.

- La durée de rotation

C'est le délai moyen entre l'admission du processus et la fin de son exécution.

- Le temps d'attente

C'est le temps moyen qu'un processus passe à attendre. Il s'obtient en soustrayant la durée d'exécution du processus de sa durée de rotation. Cette durée est indépendante de la durée d'exécution du processus lui-même.

- Le temps de réponse *Ce critère s'applique aux système interactifs*

C'est la vitesse de réaction aux interventions extérieures. Les programmes d'avant-plan doivent pour cela avoir priorité sur les tâches de fond.

- La prévisibilité

Un système qui d'habitude réagit rapidement aux commandes mais qui parfois prend un temps beaucoup plus long sera perçu comme moins stable que s'il répondait à chaque fois dans un temps comparable même s'il est globalement plus lent.

Le système semblera aussi plus convivial s'il respecte l'idée (parfois fausse) que les utilisateurs se font de la complexité des tâches.

Algorithmes d'ordonnancement *« scheduling algorithm »*

FCFS *Fist-come First-served* = Premier arrivé / Premier servi

Les jobs attendent dans une file. Le premier arrivé est admis immédiatement et s'exécute tant qu'il n'est pas bloqué ou terminé. Lorsqu'il se bloque, le processus suivant commence à s'exécuter et le processus bloqué va se mettre au bout de la file d'attente.
C'est typiquement un algorithme non préemptif.

Avantages : l'algorithme est simple (c'est une simple liste chaînée)
l'ordonnancement est équitable

Inconvénient : Le processus qui utilise davantage de temps processeur est favorisé par rapport à ceux qui font beaucoup d'appels aux entrées/sorties.

SJF *Shorted Job First* = Le job le plus court d'abord

Sera élu, le processus dont on suppose que le traitement sera le plus court. Si quatre jobs ont des durées d'exécution a, b, c et d ; le premier se termine à l'instant a, le second à l'instant a+b et ainsi de suite. La durée de rotation moyenne est de (4a + 3b + 2c + d) / 4
Le premier job exécuté contribue donc beaucoup plus que les autres à la durée moyenne. Il est donc normal d'exécuter le plus court en premier lieu. *(C'est ce qui se passe quand à la caisse d'une grande surface les clients laissent passer devant quelqu'un qui n'a qu'un article)*

Avantage : meilleure prévisibilité

Inconvénient : les jobs les plus courts sont favorisés. Si des processus courts arrivent sans cesse, les processus plus longs n'auront jamais le temps de s'exécuter

SRT - *Shorted Remaining Time* = L'algorithme du temps restant le plus court

C'est la version préemptive de l'algorithme précédent.
(Cette fois le client qui n'a qu'un article vient carrément interrompre la caissière. Non mais !)

RR *Round Robin* = L'algorithme du tourniquet

Chaque processus reçoit tour à tour un intervalle de temps appelé quantum. Au terme de ce quantum ou, si le processus s'achève ou se bloque avant cet instant, l'ordonnanceur attribue directement le processeur au processus suivant. L'algorithme est simple et équitable. C'est généralement cet ordonnancement circulaire qui est utilisé dans les systèmes à temps partagé.
La valeur du quantum est couramment fixée aux alentours de 20 à 50 ms.
Des quanta plus courts provoqueraient trop de commutations de processus et la proportion du temps consacré à ces changements de contexte deviendrait trop importante.
Si par contre, on optait pour des quanta plus longs, ce seraient les temps de réponse aux processus interactifs qui en pâtiraient.

L'ordonnancement avec priorité

Une valeur de priorité est assignée à chaque processus. Elle peut être fonction d'un ordre de préséance entre utilisateurs ou fonction des caractéristiques des processus. Un processus temps réel sera par exemple prioritaire par rapport à une tâche de fond.
La priorité peut varier dynamiquement. Exemple : pour ne pas encombrer la mémoire avec des processus qui passent le plus clair de leur temps à attendre des entrées/sorties, on leur accorde une priorité d'autant plus grande qu'ils ne consomment qu'une petite fraction de leur quantum. Supposons que la valeur du quantum est fixée à 50 ms. Un processus qui n'utilise

que 1 ms avant d'être bloqué aurait droit à une priorité de 50 ms/ 1 ms = 50 tandis qu'un processus qui se bloque au bout de 25 ms a droit à une priorité de 50/25=2 et les processus qui consomment tout le quantum ont une priorité de 1.