L'ordonnancement des processus

1. Introduction

Dans un système d'exploitation il est courant que plusieurs processus soient simultanément prêts à s'exécuter. Il faut donc réaliser un choix pour « ordonnancer », dans le temps, les processus prêts sur le processeur.

Le choix d'un algorithme d'ordonnancement dépend de l'utilisation que l'on souhaite faire de la machine et s'appuie sur les critères suivants :

- **Equité :** chaque processus doit pouvoir disposer de la ressource processeur.
- **Efficacité :** l'utilisation du processeur doit être maximale.
- **Temps de réponse :** Il faut minimiser le temps de réponse pour les utilisateurs interactifs.
- **Temps d'exécution :** Il faut minimiser le temps d'exécution pris par chaque travail exécuté.
- **Rendement :** Le nombre de travaux réalisés par unité de temps doit être maximal.

En fait plusieurs de ces critères sont mutuellement contradictoires et l'on a montré que tout algorithme d'ordonnancement qui favorise une catégorie de travaux le fait au déterminent de l'autre.

De plus, rien ne permet de connaître à l'avance les demandes en ressources de chacun des processus (E/S, mémoire, processus,…) au cours de leur exécution.

Pour assurer l'équité entre les processus, il est donc nécessaire de mettre en œuvre un mécanisme de temporisation, afin de rendre la main à l'ordonnanceur pour que celui-ci puisse déterminer si le processus courant peut continuer ou doit être suspendue au profit d'un autre processus. On effectue alors un ordonnancement avec réquisition du processeur, bien plus complexe à réaliser que le simple ordonnancement par exécution jusqu'à achèvement, car il implique la possibilité de conflits d'accès, qu'il faut prévenir au moyen de mécanismes délicats (sémaphore ou autre).

2. La stratégie PAPS (Premier Arrivé Premier Servi)

La stratégie PAPS est la plus simple à mettre en œuvre. Le traitement du processus est séquentiel : Le premier arrivé en file d'attente est le premier qui sera traité. Avec cet algorithme les processus court sont pénalisés.

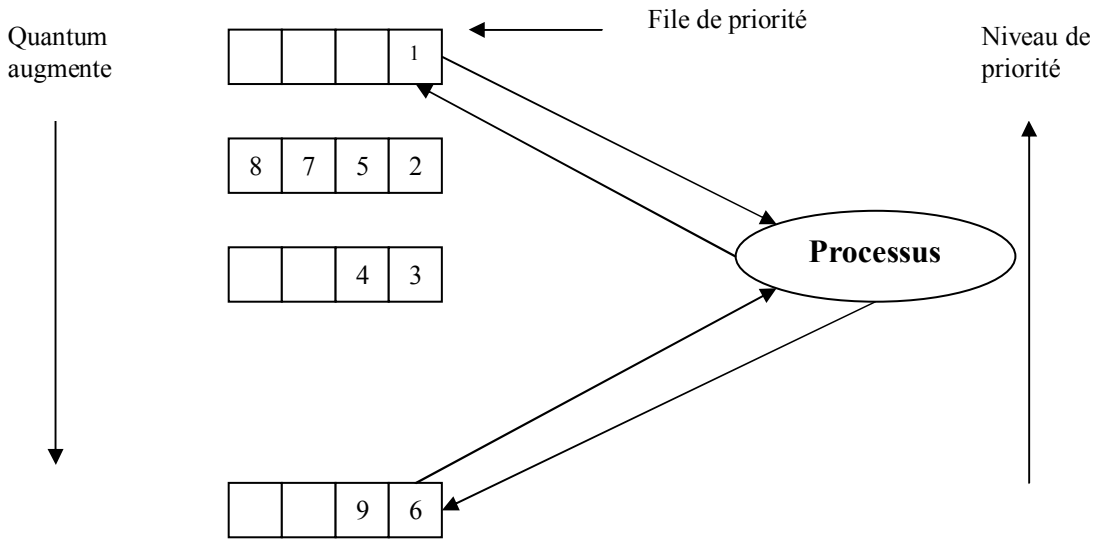
3. Ordonnancement circulaire

Le mécanisme d'ordonnancement circulaire est l'un des plus simples et plus robustes. Il consiste à attribuer à chaque processus un quantum de temps pendant lequel il a le droit de s'exécuter. Si un processus s'exécute jusqu'à l'épuisement de son quantum, le processeur est réquisitionné par l'ordonnanceur et est attribué à un autre processus.

En fait, la principale question relative à l'algorithme d'ordonnancement circulaire concerne le choix de la durée du quantum par rapport à celle d'un changement de contexte. Un quantum trop petit conduit à un gaspillage de la ressource processeur alors qu'un quantum trop grand réduit fortement l'interactivité du système lorsque de nombreux processus sont en attente d'exécution.

En pratique un quantum d'une centaine de millisecondes constitue un compromis acceptable.

4. Ordonnancement avec priorité



Exemple d'ordonnancement à plusieurs files

Dans la plus part des cas on souhaite mettre en œuvre un mécanisme de priorité afin de favoriser certaines classes de processus par rapport à d'autres.

Les critères de priorité peuvent dépendre du type de travail de l'utilisateur demandeur.
Il est souvent pratique de regrouper les processus par classe de priorité et d'utiliser l'ordonnancement par priorité entre classe et l'ordonnancement circulaire entre les processus d'une même classe.
On peut remarquer que les processus de plus faibles priorités sont les plus gros consommateurs de ressources. Afin d'améliorer l'efficacité du système il est donc possible d'attribuer au processus de plus basse priorité un quantum de temps plus grand afin de minimiser le nombre de changement de contexte.

→ Ordonnancement à plusieurs files

On travaille avec plusieurs files d'attentes. Les files d'attentes permettent d'introduire une hiérarchie entre les processus demandeur en fonction d'une priorité associée au processus et du temps déjà passé dans le système.
Les règles de fonctionnement sont les suivantes :

- Le niveau de priorité maximale (niveau 1) correspond à la file d'attente des nouveaux processus.
- A chaque file d'attente est associé un quantum de temps spécifique.
- La file la moins prioritaire a un quantum de temps plus important.
- Quand un processus atteint la fin de son quantum sans être terminé il descend d'un étage (niveau de priorité inférieur)
- Les processus d'un niveau de priorité ne peuvent accéder au processeur que si les niveaux inférieurs sont vides.
- L'apparition d'un nouveau processus dans une file d'attente de rang inférieur à celle d'origine d'un processus en cour d'exécution provoque le vidage de ce processus c.à.d la réquisition du processeur.

5. Ordonnancement du plus court d'abord

Ce type d'ordonnancement s'applique lorsque on dispose d'un ensemble de taches dont on peut connaître la durée à l'avance ; comme par exemple dans le traitement la transaction journalière bancaire. Si la file d'attente contient plusieurs taches de même priorité on minimise le temps de réponse en effectuant toujours d'abord celle ayant le temps le plus court.
Cette technique peut être appliquée au processus interactif (commande exécutée par les utilisateurs) en considérant que chaque commande tapé par l'utilisateur constitue une tache indépendante : le problème est alors de connaître le temps que prendra une commande donnée. Ceci n'est pas possible mais on peut l'estimer en se basant sur le temps des commandes précédentes :
→Une solution possible consiste à estimer le temps de la commande suivante en calculant une moyenne sur le temps des dernières commandes

effectuées, pondérées de façon à donner plus d'importance à la commande précédente. Pratiquement on utilise un coefficient a tel que $0 \le a \le 1$ tel que le temps estimé T_i^e à l'état i soit égal à :

$T_i^e = aT_{i-1}^m + (1-a)T_{(i-1)}^e$

où :

T_{i-1}^m : temps mesuré à l'étape *(i-1)*

$T_{(i-1)}^e$: temps estimé à l'étape *(i-1)*

Exemple : $a = \frac{1}{2}$

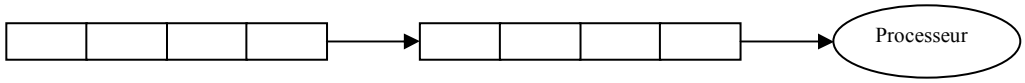
$$T_1^e = \frac{1}{2}T_0^m + \frac{1}{2}T_0^e$$

et puisque on a pas d'estimation au début on fait $T_0^e = T_0^m$

$$T_1^e = \frac{1}{2}T_0^m + \frac{1}{2}T_0^m = T_0^m$$
$$T_2^e = \frac{1}{2}T_1^m + \frac{1}{2}T_1^e = \frac{1}{2}T_1^m + \frac{1}{2}T_0^m$$
$$T_3^e = \frac{1}{2}T_2^m + \frac{1}{2}T_2^e = \frac{1}{2}T_2^m + \frac{1}{4}T_1^m + \frac{1}{4}T_0^m$$
$$T_4^e = \frac{1}{2}T_3^m + \frac{1}{2}T_3^e = \frac{1}{2}T_3^m + \frac{1}{4}T_2^m + \frac{1}{8}T_1^m + \frac{1}{8}T_0^m$$

Cette méthode de calcul, qui fait décroître le poids des mesures les plus anciennes est appelée méthode de vieillissement.

6. Ordonnancement à deux niveaux



File d'attente haut niveau
Disque dur

File d'attente bas niveau
Mémoire centrale

Tous les mécanismes d'ordonnancements considérés jusqu'à présent supposent que l'ensemble des processus était accessible en mémoire. Cependant si la mémoire physique est limitée et que l'on dispose de mécanisme de va et vient, le temps de commutation (entre deux processus) vers un processus diffère de plusieurs ordres de grandeurs, selon que ce processus est déjà situé en mémoire ou déporté sur le disque.

Pour cela on met en place un ordonnanceur à deux niveaux :

- Un ordonnanceur de bas niveau : s'occupe des processus en mémoire.
- Un ordonnanceur de haut niveau : de temps en temps (quand l'ordonnanceur de bas niveau lui alloue le processeur) échange le processus entre la mémoire et le disque.

Pour décider de migrer un processus donné depuis le disque vers la mémoire centrale ou réciproquement, l'ordonnanceur de haut niveau peut se baser sur les critères suivants :

- Le temps écoulé depuis de dernier va et vient du processus
- La quantité de temps processeur récemment consommé par le processus
- La taille du processus
- La priorité des processus